Omnis Command Reference

Omnis Software Ltd

May 2023 - Updated Oct 2023 Revision 35659

About this Manual

This manual contains a complete list of 4GL commands available in Omnis Studio, arranged in alphabetical order. See the Omnis Programming manual for further information about using the Omnis commands.

Command information

Each command has the following information, as well as the syntax, description, and an Omnis code example.

Command group	Flag affected	Reversible	Execute on Client	Platform
Functional group,e.g. "Constructs"	Whether or not (YES/NO) the command sets the flag when it executes; if the command executes successfully the flag is set to True, if it fails the flag is set to False.	Whether or not (YES/NO) the command is reversed when it is executed within a reversible block; see *Begin reversible block* command.	Whether or not (YES/NO) the command can be executed in a client method in the JavaScript Client; see also Client commands	Which p comma includin macOS; comma platform

Command Groups

In versions of Omnis Studio prior to version 10.x the commands were arranged in groups in the Method Editor, but the command groups no longer appear in the Code Editor. The commands are listed here in the same functional groups for your convenience only.

Calculations	Classes	Constructs	Debugger
Error handlers	Events	External commands	Externals
Libraries	Message boxes	Methods	Operating system
Parameters and variables	Reports and Printing	SQL Object Commands	Text
Threads			

The following commands apply to desktop apps only, and should not be used in web or mobile apps.

Changing data	Clipboard	Data files	Data management
Enter data	Exchanging data	Fields	Files
Finding data	Importing and Exporting	List lines	Lists
Menus	Omnis environment	Report destinations	Report parameters
Searches	Sort fields	Tasks	Toolbars

Client Commands

The following commands can be executed in a client method in the JavaScript Client.

Command
Comment
Begin text block
Break to end of switch
asof 35949 Breakpoint
Calculate
Case
Default
Do
Do inherited
Do method
Else
Else If calculation
Else If flag false
Else If flag true
End For
End If
End Switch
End text block
End While
For field value
Get text block
If calculation
If flag false
If flag true
JavaScript:
Jump to start of loop
OK message
On
On default
Quit event handler
Quit method
Repeat
Send to trace log
Set reference
Sound bell
Switch
Text:
Until calculation
Until flag false
Until flag true
While calculation
While flag false
While flag true

Error Codes

FileOps Error Codes

Web Error Codes

Obsolete Commands

There were several commands or command groups marked as 'Obsolete' in versions of Omnis Studio prior to Studio 10.0 and these have been removed from Omnis Studio and will be commented out in converted libraries: for the benefit of existing users, the obsolete commands are listed here: Obsolete Commands

Command Filters

The commands in Omnis perform many different functions, including many legacy features that are no longer required for creating web and mobile apps using the JavaScript Client. There is a filter mechanism in the Method Editor to filter the list of commands that are displayed in the **Code Assistant** help list, primarily to remove any old commands, including those for managing Omnis datafiles.

Note you can still use the excluded commands in your code, and methods in converted libraries using these commands will continue to work – the filters just hide the commands from the Code Assistant help list.

The command filter is set under the **Filter Commands** submenu in the **Modify** menu in the Method Editor: note this is only visible when the cursor is in Code Editor, while editing a line of code. The **Exclude Old Commands** filter is enabled by default, which excludes over 200 old commands, plus there are other filters available that exclude smaller subsets of commands. You can disable the current filter using the **No Filter** option, in which case all the commands available in Omnis will be shown in the Code Assistant help list.

Copyright info

Lists

Commands

The **Lists** group of commands are no longer visible in the Code Assistant in the Code Editor (they will not appear when you type the first few characters), although they are still present in Studio 11 and will continue to function in legacy code. You can show these commands by disabling the appropriate Command Filter in the Modify menu in the Code Editor.

You should use the equivalent methods where available, such as $define() instead of *Define list*, $search() instead of *Search list*, $sort() instead of *Sort list*, and so on, to manipulate the contents of list variables.

Build list columns list	Build list from file	Clear list	Copy list definition
Define list	Define list from SQL class	Merge list	Redefine list
Search list	Set current list	Sort list	Swap lists

Menus

Commands

These commands are for desktop apps using menu classes only, not web or mobile apps.

Build installed menu list	Build menu list	Check menu line	Disable all menus and toolbars
Disable menu line	Enable all menus and toolbars	Enable menu line	Install menu
Popup menu	Popup menu from list	Redraw menus	Remove all menus
Remove final menu	Remove menu	Replace standard Edit menu	Replace standard File menu
Standard menu command	Test for menu installed	Test for menu line checked	Test for menu line enabled
Uncheck menu line			

Message boxes

Commands

Close working message *	No/Yes message	OK message	Prompt for input *
Redraw working message *	Sound bell	Working message *	Yes/No message

**These commands are for desktop apps only, not web or mobile apps.*

Methods

Commands

Cancel async method *	Clear method stack	Clear timer method	Do async method *
Do code method	Do method	Optimize method	Quit all if canceled *
Quit all methods *	Quit method	Quit Omnis	Set timer method

**These commands are for desktop apps only, not web or mobile apps.*

Obsolete Commands

Some of the obsolete commands have been removed from this version: these commands were marked with "OBSOLETE COMMAND" and appeared in the 'Obsolete commands...' group in the Command list in pre-Studio 10.x versions. The converter in Studio 10.x will comment out these commands wherever they appear in your code, and a record of the conversion process is added to a log file in the /logs/conversion folder.

* The *Call method OBSOLETE COMMAND* is not commented out, but is converted to *Do code method* using the same parameter as the old command.

The *Translate input/output* command is now obsolete and will be commented out in your converted code.

Autocommit OBSOLETE COMMAND

Begin SQL script OBSOLETE COMMAND

Build list from select table OBSOLETE COMMAND

Build list of event recipients OBSOLETE COMMAND

Call method OBSOLETE COMMAND * (converted to Do code method)

Cancel event recipient OBSOLETE COMMAND

Cancel publisher OBSOLETE COMMAND

Cancel subscriber OBSOLETE COMMAND

Close client import file OBSOLETE COMMAND

Close cursor OBSOLETE COMMAND

Commit current session OBSOLETE COMMAND

Declare cursor for OBSOLETE COMMAND

Delete client import file OBSOLETE COMMAND

Describe cursors OBSOLETE COMMAND

Describe database OBSOLETE COMMAND

Describe results OBSOLETE COMMAND

Describe server table OBSOLETE COMMAND

Describe sessions OBSOLETE COMMAND

Disable automatic publications OBSOLETE COMMAND

Disable automatic subscriptions OBSOLETE COMMAND

Disable receiving of Apple events OBSOLETE COMMAND

Enable automatic publications OBSOLETE COMMAND

Enable automatic subscriptions OBSOLETE COMMAND

Enable receiving of Apple events OBSOLETE COMMAND

End SQL script OBSOLETE COMMAND

Execute SQL script OBSOLETE COMMAND

Fetch current row OBSOLETE COMMAND

Fetch first row OBSOLETE COMMAND

Fetch last row OBSOLETE COMMAND

Fetch next row OBSOLETE COMMAND

Fetch previous row OBSOLETE COMMAND

Get SQL script OBSOLETE COMMAND

Logoff from host OBSOLETE COMMAND

Logon to host OBSOLETE COMMAND

Make file class from server table OBSOLETE COMMAND

Make schema from server table OBSOLETE COMMAND

Map fields to host OBSOLETE COMMAND

Open client import file OBSOLETE COMMAND

Open cursor OBSOLETE COMMAND

Open desk accessory OBSOLETE COMMAND

Perform SQL OBSOLETE COMMAND

Prepare current cursor OBSOLETE COMMAND

Prompt for event recipient OBSOLETE COMMAND

Prompt for word server OBSOLETE COMMAND

Publish field OBSOLETE COMMAND

Publish now OBSOLETE COMMAND

Quit cursor(s) OBSOLETE COMMAND

Reset cursor(s) OBSOLETE COMMAND

Retrieve rows to file OBSOLETE COMMAND

Rollback current session OBSOLETE COMMAND

Send core event OBSOLETE COMMAND

Send core event with return value OBSOLETE COMMAND

Send database event OBSOLETE COMMAND

Send finder event OBSOLETE COMMAND

Send to publisher OBSOLETE COMMAND

Send word services event OBSOLETE COMMAND

Server specific keyword OBSOLETE COMMAND

Set batch size OBSOLETE COMMAND

Set character mapping OBSOLETE COMMAND

Set client import file name OBSOLETE COMMAND

Set current cursor OBSOLETE COMMAND

Set current session OBSOLETE COMMAND

Set database version OBSOLETE COMMAND

Set event recipient OBSOLETE COMMAND

Set hostname OBSOLETE COMMAND

Set password OBSOLETE COMMAND

Set publisher options OBSOLETE COMMAND

Set SQL blob preferences OBSOLETE COMMAND

Set SQL script OBSOLETE COMMAND

Set SQL separators OBSOLETE COMMAND

Set subscriber options OBSOLETE COMMAND

Set transaction mode OBSOLETE COMMAND

Set username OBSOLETE COMMAND

SQL: OBSOLETE COMMAND

Start session OBSOLETE COMMAND

Subscribe field OBSOLETE COMMAND

Subscribe now OBSOLETE COMMAND

Translate input/output

Use event recipient OBSOLETE COMMAND

Omnis environment

Commands

These commands are for desktop apps only, not web or mobile apps.

Set 'About...' method	Set Omnis window title	Show 'About...' window	Show Omnis maximized
Show Omnis minimized	Show Omnis normal	Test if running in background	

Operating system

Commands

Context help *	Launch program	Start program maximized *	Start program minimized *
Start program normal *	Test for program open	Test if file exists	

**These commands are for desktop apps only, not web or mobile apps.*

Parameters and variables

Commands

Clear class variables			

Report destinations

Commands

Close print or export file	Set print or export file name		

These commands are for desktop apps only, not web or mobile apps.

Close port	Prompt for destination	Prompt for port name	Prompt for print or export file
Send to a window field	Send to clipboard	Send to DDE channel	Send to file
Send to page preview	Send to port	Send to printer	Set port name
Set port parameters			

Report parameters

Commands

Prompt for page setup	Set bottom margin	Set export format	Set label width
Set labels across page	Set left margin	Set lines per page	Set page width
Set record spacing	Set repeat factor	Set report main file	Set report main list
Set right margin	Set top margin		

Reports and Printing

Commands

Begin print job	End print job	Load page setup	Prepare for print

Print record	Print report	Print report from disk	Print report from memory
Select printer	Set report name	Transmit text to port	Transmit text to print file

Searches

Commands

The Searches group of commands are no longer visible in the Code Assistant in the Code Editor (they will not appear when you type the first few characters), although they are still present in Studio 11 and will continue to function. You can show these commands by disabling the appropriate Command Filter in the Modify menu in the Code Editor.

These commands are for desktop apps using Omnis data files and Search classes only, not web or mobile apps.

Build search list	Clear search class	Reinitialize search class	Set search as calcula-tion
Set search name	Test data with search class		

Sort fields

Commands

The Sort Fields commands are no longer visible in the Code Assistant in the Code Editor (they will not appear when you type the first few characters), although they are still present in Studio 11 and will continue to function. You can show these commands by disabling the appropriate Command Filter in the Modify menu in the Code Editor.

These commands are for desktop apps using Omnis data files and sort fields only, not web or mobile apps.

Clear sort fields	Set sort field		

SQL Object Commands

Commands

Begin state-ment	End state-ment	Get state-ment	Sta:

Tasks

Commands

These commands are for desktop apps using Task classes only, not web or mobile apps.

Close task instance	Open task instance

Text

Commands

Begin text block Text:	End text block	Get text block	Line:

Threads

Commands

Begin critical block Yield to other threads	End critical block	Start server	Stop server

Toolbars

Commands

These commands are for desktop apps using Toolbar classes only, not web or mobile apps.

Hide docking area Show docking area	Install toolbar	Redraw toolbar	Remove toolbar

Windows

Commands

These commands are for desktop apps using Window classes only, not web or mobile apps.

Bring window instance to front	Build open window list *	Build window list *	Close all windows
Close other windows	Close top window	Close window instance	Maximize window instance
Minimize window instance Test for window open	Open window instance	Print top window	Set top window title

*These commands are no longer visible in the Code Assistant in the Code Editor (they will not appear when you type the first few characters), although they are still present in Studio 11 and will continue to function. You can show these commands by disabling the appropriate Command Filter in the Modify menu in the Code Editor.

List lines

Commands

The **List Lines** group of commands are no longer visible in the Code Assistant in the Code Editor (they will not appear when you type the first few characters), although they are still present in Studio 11 and will continue to function in legacy code. You can show these commands by disabling the appropriate Command Filter in the Modify menu in the Code Editor.

You should use the equivalent methods where available, such as $add() instead of *Add line to list*, to manipulate the contents of list variables.

Add line to list	AND selected and saved	Clear line in list	Delete line in list
Delete selected lines	Deselect list line(s)	Go to next selected line	Insert line in list
Invert selection for line(s)	Load from list	OR selected and saved	Replace line in list
Restore selection for line(s)	Save selection for line(s)	Select list line(s)	Set final line number
Swap selected and saved	Test if list line selected	XOR selected and saved	

 # Comment

Command group	Flag affected	Reversible	Execute on client
Constructs	NO	NO	YES

Syntax

*message*

Description

This command allows you to add comments to your code. You can either add a new comment, or you can “comment out” existing lines in your code.

To enter a new comment on an empty line, you can type # and then the comment text, with or without a space after the #. (You can also type ; to create a new comment, but the comment is marked with #, since semicolon was used for comments in previous versions).

To enter an inline comment, type “<space>##” at the end of a code line, and then enter the comment text. Inline comments are positioned over on the right of the code entry area: they are left-tab aligned according to a tab which is indicated by a small marker at the top of the code entry area: you can drag this marker to reset the tab position.

Note that the Sta:, Text: and JavaScript: commands do not allow inline comments.

To “comment out” lines of code, i.e. to stop the code executing, select the method line or multiple lines and press Ctrl+/. Use the same keypress to uncomment previously commented out lines of code (in this case, the comment text must be a valid command to be uncommented).

```
# here are some comments
# variable delay set by lDelay

# adjust Until calculation to increase/decrease delay
Calculate lCount as 1
Repeat       ## this is an in-line comment
  Calculate lCount as lCount+1
Until lCount>=lDelay*10
```

Accept advise requests

Command group	Flag affected	Reversible	Ex
Exchanging data	NO	YES	NO

Syntax

Accept advise requests ([*Accept*])

Options

Accept	If specified, the mode identified by the command is enabled

Description

DDE command, Omnis as server. This command enables or disables responses to a request Advise message from a client. With the *Accept* check box selected, Omnis will respond to an Advise request message specifying a valid field name by repeatedly sending the field value to the client at appropriate times. If the *Accept* option is unchecked, all conversations with Advises in force will be terminated unless the command is part of a reversible block.

Example

```
Accept advise requests (Accept)
```

Accept commands

Command group	Flag affected	Reversible	Ex
Exchanging data	NO	YES	NO

Syntax

Accept commands ([*Accept*])

Options

Accept	If specified, the mode identified by the command is enabled

Description

DDE command, Omnis as server. This command determines whether Omnis will accept commands from the client program. When **Accept commands** is in force, Omnis will respond to a DDE EXECUTE message by attempting to execute a command string sent by the client program. All conversations are terminated when you close your Omnis library.

Example

```
Accept advise requests (Accept)
Accept commands (Accept)
```

Accept field requests

Command group	Flag affected	Reversible	Ex
Exchanging data	NO	YES	NO

Syntax

Accept field requests ([*Accept*])

Options

Accept	If specified, the mode identified by the command is enabled

Description

DDE command, Omnis as server. This command enables or disables responses to a request for field values issued by a client application. With the *Accept* option selected, Omnis will respond to a Request message specifying a valid field name by sending the field value to the client program. Values are taken from the current record buffer. Values are only sent when Omnis is in enter data mode or when no methods are running.

Example

```
Accept advise requests (Accept)
Accept commands (Accept)
Accept field requests (Accept)
```

Accept field values

Command group	Flag affected	Reversible	Ex
Exchanging data	NO	YES	NO

Syntax

Accept field values ([*Accept*])

Options

Accept	If specified, the mode identified by the command is enabled

Description

DDE command, Omnis as server. This command determines whether Omnis is able to receive data from a client via a DDE POKE message. With the *Accept* option selected, Omnis will respond to a Poke message specifying a valid field or variable name, by setting the value of that field to the value transmitted by the client program. Values are stored in the current record buffer and, if the relevant field is on the top window, that window is redrawn.

Field values are only accepted when Omnis is in enter data mode, Prompted find, or when no methods are running. All conversations are terminated when you close your Omnis library.

Example

```
Accept advise requests (Accept)
Accept field values (Accept)
```

Add line to list

Command group	Flag affected	Reversible	Execute on client
List lines	YES	NO	NO

Syntax

Add line to list {*line-number* (*values*) {default is end of list}}

Deprecated Command

This command has been deprecated and is no longer visible in the Code Assistant in the Code Editor (it will not appear when you type the first few characters), although it is still present in Omnis Studio and will continue to function if used in legacy code. You can show this command by disabling the appropriate Command Filter in the **Modify** menu in the Code Editor.

Description

This command adds a new line to the current list using the current field values in the CRB or values you specify in the list of values. Any conversions required between data types are carried out automatically. The flag is cleared if the line cannot be added, either because the maximum number of lines in the list or the memory limits have been exceeded.

You can specify the line number at which the new line is inserted, otherwise the line is added to the end of the list. If the line number you specify in the command line is empty or evaluates to zero, the new line is added to the end of the list. If too few values are specified, the other columns are left empty; if too many values are specified, the extra values are ignored. When you supply a comma-separated list of values, the values in the CRB are ignored.

```
# Create a fixed list of string and numeric data
Set current list lMyList
Define list {lName,lAge}
Add line to list {('Fred',10)}
Add line to list {('George',20)}

# Insert the values of the variables lName and lAge to lMyList at line 1
Calculate lName as 'Harry'
Calculate lAge as 22
Add line to list {1 (lName,lAge

# If no values are defiened, the current values of the variables
# used in the Define List are added
Add line to list

# Alternatively, you can use the $add() method to add lines to your list
Do lMyList.$define(lName,lAge)
Do lMyList.$add('Fred',10)
Do lMyList.$add('George',20)

# You can also use the $addbefore() and $addafter() methods to add
# lines at a specific position in the list
Do lMyList.$addbefore(1,'Harry',22)
```

Advise on find/next/previous

Command group	Flag affected	Reversible	Ex
Exchanging data	NO	YES	NO

Syntax

Advise on find/next/previous ([*Accept*])

Options

Accept	If specified, the mode identified by the command is enabled

Description

DDE command, Omnis as server. This command determines when Omnis is permitted to send requested Advise messages to the client program. When Advise requests have been received from a client, the *Set server mode* command determines when Omnis is permitted to send field values that have changed. In addition to the *Set server mode* options, the three commands **Advise on Find/next/previous**, *Advise on OK*, and *Advise on Redraw* let you toggle individual options on or off. **Advise on Find/next/previous** lets you control this particular option without affecting the other two.

Example

```
Advise on find/next/previous (Accept)
```

Advise on OK

Command group	Flag affected	Reversible	Ex
Exchanging data	NO	YES	NO

Syntax

Advise on OK ([*Accept*])

Options

Accept	If specified, the mode identified by the command is enabled

Description

DDE command, Omnis as server. This command determines when Omnis is permitted to send requested Advise messages to the client program. When Advise requests have been received from a client, the *Set server mode* command determines when Omnis is permitted to send field values that have changed. In addition to the *Set server mode* options, the three commands *Advise on Find/next/previous*, **Advise on OK**, and *Advise on Redraw* let you toggle individual options on or off. The **Advise on OK** command lets you control this particular option without affecting the other two.

Example

```
Advise on OK (Accept)     ## enable advise on OK
Advise on OK     ## disable advise on OK
```

Advise on redraw

Command group	Flag affected	Reversible	Ex
Exchanging data	NO	YES	NO

Syntax

Advise on redraw ([*Accept*])

Options

Accept	If specified, the mode identified by the command is enabled

Description

DDE command, Omnis as server. This command determines when Omnis is permitted to send requested Advise messages to the client program. When Advise requests have been received from a client, the *Set server mode* command determines when Omnis is permitted to send field values that have changed. In addition to the *Set server mode* options, the three commands *Advise on Find/next/previous, Advise on OK,* and **Advise on redraw** let you toggle individual options on or off. The **Advise on redraw** command lets you control this particular option without affecting the other two.

Example

```
Advise on redraw (Accept) ## enable advise on redraw
Advise on redraw ## disable advise on redraw
```

AND selected and saved

Command group	Flag affected	Reversible	Execute on client
List lines	YES	NO	NO

Syntax

AND selected and saved ([*All lines*]) {*line-number* (*calculation*)}

Options

All lines	If specified, the command affects all the lines in the list

Deprecated Command

This command has been deprecated and is no longer visible in the Code Assistant in the Code Editor (it will not appear when you type the first few characters), although it is still present in Omnis Studio and will continue to function if used in legacy code. You can show this command by disabling the appropriate Command Filter in the **Modify** menu in the Code Editor.

Description

This command performs a logical AND of the Saved selection with the Current selection. You can specify a particular line in the list by entering either a number or a calculation. The *All lines* option performs the AND for all lines of the current list.

To allow sophisticated manipulation of data via lists, a list can store two selection states for each line; the “Current” and the “Saved” selection. The Current and Saved selections have nothing to do with saving data on the disk; they are no more than labels for two sets of selections. The lists may be held in memory and never saved to disk: they will still have a Current and Saved selection state for each line but they will be lost if not saved. When a list is stored in the data file, both sets of selections are stored.

The list data structure contains the column definitions, the field values for each line of the list, the current selected status and saved selected status for each line, *LIST.$line*, *LIST.$linecount* and *LIST.linemax*.

The **AND selected and saved** command performs a logical AND on the saved and current state, and puts the result into the Current selection. Hence, for a particular line, if both the Current and Saved states are selected, the Current state remains selected, but if either or both states are deselected, the resulting Current state will become deselected.

Saved State	Current State	Resulting Current State
Selected	Selected	Selected
Deselected	Selected	Deselected
Selected	Deselected	Deselected
Deselected	Deselected	Deselected

Example

```
# Line 3 remains selected as it is the only line selected
# when both the 'Save selection for line(s)' and
# 'AND selected and saved' commands are used
Set current list lMyList
Define list {lCol1}
For lCol1 from 1 to 6 step 1
  Add line to list {lCol1}
End For
Select list line(s) (All lines)
Save selection for line(s) (All lines)
Deselect list line(s) (All lines)
Select list line(s) {3}
AND selected and saved (All lines)
```

Begin critical block

Command group	Flag affected	Reversible	Execute on client	P
Threads	NO	NO	NO	A

Syntax

Begin critical block

Description

Begin critical block is only applicable to the multithreaded server. It marks the start of a *critical block*, namely a section of code which needs to execute in single threaded mode without allowing other client methods to execute. You use *End critical block* to mark the end of a critical block.

One example of when you should use a critical block is as follows. Class variables are shared by all clients. Simple atomic operations, such as the direct assignment of a value to a class variable are safe. Other operations, such as when a method call is involved, could cause problems, because the method call might be interrupted by another thread. To avoid this, use a critical block.

Example

```
Begin critical block
  Calculate cClassVar as $cinst.$getvalue()
End critical block
```

Begin print job

Command group	Flag affected	Reversible
Reports and Printing	YES	NO

Syntax

Begin print job ([*Send to PDF*])

Options

Send to PDF	If specified, the print job sends its reports to a single PDF file rather than a printer document

Description

This command defines the beginning of an Omnis print job which is ended by the command *End print job*. Only one print job can be started at any time: you cannot nest **Begin print job** commands.

If printing is already in progress, **Begin print job** returns an error and sets the flag to false. It also returns an error if it cannot set up the printer, or open the printer document; again, it sets the flag to false in this case.

Begin print job sets the flag to true if it succeeds. It automatically sets the report destination to the printer and closes the report destination selection window if it is open.

Each report is printed in the same way as if it were in an individual document. If you print two reports in a job, then page numbering starts at 1 for each report.

You cannot change the page setup while a print job is in progress, although Omnis does not try to enforce this, as it will probably cause an OS error (and abnormal termination of printing) if you do.

The **Begin print job** and *End print job* commands only apply to reports sent to a printer, via the printer report destination.

Example

```
# Create a print job and send 2 reports to the printer
Begin print job
Set report name rMyReport
Print report
Set report name rMyReport2
Print report
End print job
```

Begin reversible block

Command group	Flag affected	Reversible	Execute on client
Constructs	NO	NO	NO

Syntax

Begin reversible block

Description

This command begins a reversible block of commands. All reversible commands enclowsed within the commands Begin reversible block/End reversible block are reversed when the method containing this block finishes. However, a reversible block in the $construct() method of a window class reverses when the window is closed not when the method is terminated as is normally the case. Omnis always steps backwards through a reversible block of commands, thus the first command is reversed last.

Reversible blocks let you create subroutines that restore the values of variables, the current record buffer, and so on, to their previous state when the method terminates. Most commands are reversible: those that are not usually involve an irreversible action such as changing the data in an Omnis data file or running another program. Methods called from within a reversible block are not reversed.

Example

```
# A method can contain more than one block of reversible commands. In this case,
# commands contained within all the blocks are reversed when the method terminates.
# All the commands in the following example are reversed when the method containing
# the block is finished
Begin reversible block
  Disable menu line mMyMenu/5
  Set current list iMyList
  Build open window list (Clear list )
  Calculate iVar as 0
  Open window instance wMyWindow
End reversible block

# When this block is reversed:
# The window instance wMyWindow is closed
# iVar returns to its former value
# iMyList is restored to its former contents and definition
# The current list is set to the former value
# Menu line 5 is enabled
# The following method hides fields Entry1 and Entry2 and installs the menu mCustomers
Begin reversible block
  Hide fields {Entry1,Entry2}
  Install menu mCustomers
End reversible block
OK message (Icon) {MCUSTOMERS is now visible}

# When this method ends, first MCUSTOMERS is removed, then the fields are shown.
# In the following example, the current list is iMyList
Begin reversible block
  Set current list iMyList2
  Define list {fAccounts.Code,fAccounts.Surname,fAccounts.Balance}
  Set main file {fAccounts}
  Build list from select table
  Enter data
End reversible block

# When this method terminates and the command block is reversed, the Main file is reset,
# the former list definition is restored and the current list is restored to iMyList.
```

Begin statement

Command group	Flag affected	Reversible
SQL Object Commands	NO	NO

Syntax

Begin statement ([*Carriage return*][,*Linefeed*])

Options

Carriage return	If specified, the command appends a carriage return, after it appends each line of the statement
Linefeed	If specified, the command appends a line feed, after it appends each line of the statement

Description

This command defines the start of a block of SQL statements and text to be stored in the SQL buffer for the current method stack. The current content of the SQL buffer is cleared when you execute this command. The *End statement* command defines the end of the block. The lines are not checked by Omnis in any way and must be valid SQL in order for the server to be able to use them. To use the SQL buffer, you call the *$prepare* or *$execdirect* method of a SQL statement object, passing no parameters.

The *Carriage return* option causes Omnis to insert a carriage return character between each line of the SQL statement. The *Linefeed* option causes Omnis to insert a linefeed character between each line of the SQL statement. If you select both *Carriage return* and *Linefeed*, then Omnis inserts a carriage return followed by a linefeed. If you select neither option, Omnis separates the statement lines with a space. One example of when you would use these options, is when you use *Begin statement*, *Sta:, End statement,* and *$execdirect*, to add a stored procedure to the database. This makes the procedure more readable when you view it.

Example

```
# Open a multi-threaded omnis sql connection to
# the datafile mydatafile and create a statement to
# select rows from the table Customers
Calculate lHostname as con(sys(115),'mydatafile.df1')
Do iSessObj.$logon(lHostname,'','','MYSESSION')
Do iSessObj.$newstatement('MyStatement') Returns lStatObj

Begin statement
  Sta: Select * From Customers
  Sta: Where Cust_ID > 100
End statement
Do lStatObj.$execdirect()
Do lStatObj.$fetch(lMyList,kFetchAll)
```

Begin text block

Command group	Flag affected	Reversible	Execute on client	P
Text	NO	NO	YES	A

Syntax

Begin text block ([*Keep current contents*][,Carriage return][,Linefeed])

Options

Keep current contents	If specified, the command keeps the current contents of the text block rather than setting it to empty
Carriage return	If specified, Omnis appends a carriage return after it appends the text from a Line: command in the block. If the Carriage return and Linefeed options are both omitted then Omnis appends a platform newline after each Line: command
Linefeed	If specified, Omnis appends a line feed after it appends the text from a Line: command in the block. If the Carriage return and Linefeed options are both omitted then Omnis appends a platform newline after each Line: command

Description

This command defines the start of a block of text to be stored in the text buffer for the current method stack. The **Begin text block** command clears the text buffer by default, and adds the text in subsequent Line: and *Text:* commands to the text buffer. However, you can keep the current contents of the buffer by checking the *Keep current contents* option, in which case text is appended to current text in the buffer. You build the text block using the Line: and *Text:* commands, which support leading and trailing spaces and can contain square bracket notation. The *Carriage return* and *Linefeed* options specify the line delimiter added after each Line: command; if you omit both of these options, Omnis adds the platform specific newline character sequence after each Line: command. The *End text block* command defines the end of the text block, and you can return the contents of the text buffer using the *Get text block* command.

Example

```
Begin text block
  Text: Thought for the day: (Carriage return)
  Text: If a train station is where the train
  Text: stops, what is a work station?
End text block
Get text block lTextString
OK message {[lTextString]}
```

Break to end of loop

Command group	Flag affected	Reversible	Execute on client
Constructs	NO	NO	YES

Syntax

Break to end of loop

Description

This command terminates a Repeat, While or For loop, passing control to the command following the Until, End While or End For command. An If command is usually placed before the Break to end of loop to determine the condition under which a break occurs.

You cannot use the *Break to end of loop* command to break out of a Switch construct. In this case, you must use the Break to end of switch command.

Example

```
# loop until user replies yes to yes/no message or lCount=100
While lCount<-100
  Yes/No message {Break to end of loop ?}
  If flag true
    Break to end of loop
  End If
  Calculate lCount as lCount+1
End While
```

Break to end of switch

Command group	Flag affected	Reversible	Execute on client	P
Constructs	NO	NO	YES	A

Syntax

Break to end of switch

Description

This command causes Omnis to jump out of the current Case statement (i.e. terminate the Case before the end of Case is reached), and resume method execution after the End Switch command. You use it in conjunction with the Switch and Case commands.

Example

```
# If lCount equals 1 or 2 the ok message following the Break to end of switch never gets shown
Switch lCount
  Case 1
    OK message {lCount equals 1}
    Break to end of switch
    OK message {I never run}
  Case 2
    OK message {lCount equals 2}
    Break to end of switch
    OK message {I never run}
  Default
    OK message {lCount not equal to 1 or 2}
End Switch
```

Breakpoint

Command group	Flag affected	Reversible	Execute on client
upto 35948 Debugger	NO	NO	NO
asof 35949 Debugger	NO	NO	YES (see note below)

Syntax

Breakpoint {message}

Description

This command places a breakpoint at a command line in a method where you want to stop execution, to check your coding for example. You can include a message with the command which is displayed in the debug window when the break occurs. The command does nothing in the Runtime version of Omnis.

When Omnis encounters a breakpoint the debugger is opened with the current method loaded and the Breakpoint command line highlighted. You can examine the value of fields and variables by right button/Ctrl-clicking on the field or variable name.

Following a breakpoint you can continue method execution by clicking the **Go** button or by using **Step** or **Trace** mode.

The Breakpoint command is ignored if executed on a thread running on a multi-threaded Omnis Server.

NOTE: The *Breakpoint* command can be used in client-executed methods to set a 'hard' breakpoint in the code, but note that this will only be hit if the web browser developer tools are open. It will then break into the browser's debugger, in the JavaScript code which was generated from your client-executed method. The browser dev tools can usually be opened using the F12 key.

Example

```
# hit breakpoint when line 5 is processed so we can check the values of lMyList columns
For lMyList.$line from 1 to lMyList.$linecount step 1
  If lMyList.$line=5
    Do lMyList.$loadcols()
    Breakpoint {check lMyList columns}
  End If
End For
```

Bring window instance to front

Command group	Flag affected	Reversible	Execute on client
Windows	NO	NO	NO

Syntax

Bring window instance to front window-instance-name

Description

Example

```
# Bring the window instance wMyWindow
# to the front if it is already open
Test for window open {wMyWindow}
If flag true
  Bring window instance to front wMyWindow
Else
  Open window instance wMyWindow
End If
```

Build export format list

Command group	Flag affected	Reversi
Importing and Exporting	YES	YES

Syntax

Build export format list ([*Clear list*])

Options

Clear list	If specified, the command empties the current list, and defines it to have a single hash variable column, before executing

Description

This command builds a list containing the name of each export format. The list is built in the current list for which you must define a single column to contain the export format.

The *Clear list* option clears the current list and redefines it to include only the #S4 field. With this option, the command becomes reversible.

Example

```
Set current list lExportFormatList
# clear list option defines the list as a single column #S4
Build export format list (Clear list )
```

Build externals list

Command group	Flag affected	Reversible	Execute on client
Externals	YES	YES	NO

Syntax

Build externals list ([*Clear list*])

Options

Clear list	If specified, the command empties the current list, and defines it to have a single hash variable column, before executing

Description

This command builds a list of the external routines in the external folder. The list is placed in the current list for which you must define the following columns

Col 1 (Character)	Col 2 (Character	Col 3 (Number)	Col 4 (Character)
File name	Routine name	Routine index or ID	Routine type

The *Clear list* option clears the current list. The command becomes reversible with this option.

Example

```
Begin reversible block
  Set current list iExtList
End reversible block
Define list {iExtName,iExtRoutine,iExtRoutineIndex,iExtRoutineType}
Build externals list
```

Build field names list

Command group	Flag affected	Reversible	Execute on client	P
Files	YES	YES	NO	A

Syntax

Build field names list ([Clear list][,Full names]) {file-name}

Options

Clear list	If specified, the command empties the current list, and defines it to have a single hash variable column, before executing
Full names	If specified, names in the list are prefixed with their file class name

Description

This command builds a list of field names for the specified file class in the current list. You must specify the following columns in the current list.

Column 1 (Character)	Column 2 (Character)	Column 3 (Character)
Field name	Field type and length	Description; for index fields only

When you use the Clear list option you get column 1 only defined as #S5. With this option the command becomes reversible. The flag is cleared if the value of LIST.$linemax prevents a complete list from being built.

The Full names option creates a list in which the fields are prefixed with the file class name, for example, PO_DATE becomes FPORDERS.PO_DATE.

Example

```
# Build a list of the field names in the file class fAccounts
Set current list lFieldList
Build field names list (Clear list ,Full nam) {fAccounts}

# alternatively $makelist can be used
Do $files.fAccounts.$objs.$makelist($ref.$name) Returns lFieldList
```

Build file list

Command group	Flag affected	Reversible	Execute on client	P
Files	YES	YES	NO	A

Syntax

Build file list ([Clear list])

Options

Clear list	If specified, the command empties the current list, and defines it to have a single hash variable column, before executing

Description

This command builds a list containing the name of each file class in the current library. The list is built in the current list for which you must specify the following columns.

Column 1 (Character)	Column 2 (Character)
File name	Description for file (if you have entered one)

When you use the *Clear list* option you get column 1 only defined as #S5. With this option the command becomes reversible, that is, the original contents of the list are restored. The flag is cleared if the number of lines in the list exceeds LIST.$linemax.

Example

```
# Build a list of file classes in the current library
Set current list lFileList
Build file list (Clear list )
# alternatively $makelist can be used
Do $files.$makelist($ref.$name) Returns lFileList
```

Build indexes

Command group	Flag affected	Reversible	Execute on client	P
Data management	YES	NO	NO	A

Syntax

Build indexes {*file-name*}

Description

This command rebuilds all the indexes for the specified file which have been dropped with the Drop indexes command. Drop indexes deletes all the indexes for the specified file apart from the sequence number index. **Build indexes** checks that all the indexes defined in the file class actually exist in the data file and builds those which are not there. This command does not build any indexes which already exist even if they are in a damaged state.

If the specified file name does not include a data file name as part of the notation, the default data file for that file is assumed. If the file is closed or memory-only, the command does not execute and returns flag false.

If you are not running in single user mode, this command automatically tests that only one user is using the data file (the command fails with the flag false if this is not true), and further users are prevented from logging onto the data until the command completes.

If a working message with a count is open while the command is executing, the count will be incremented at regular intervals. The command may take a long time to execute, and it is not possible to cancel execution even if a working message with cancel box is open.

The flag is set if at least one index is successfully rebuilt. Note that the command is not reversible.

Example

```
Do not flush data
Drop indexes {fCustomers}
Repeat
  Working message {Building indexes...}
  Build indexes {fCustomers}
Until flag true
```

Build installed menu list

Command group	Flag affected	Reversible	Execute on client	P
Menus	YES	YES	NO	A

Syntax

Build installed menu list ([Clear list])

Options

Clear list	If specified, the command empties the current list and defines it to have a single hash variable column before executing

Description

This command builds a list containing the name of all menu instances on the main Omnis menu bar, starting from the left. All the standard Omnis menus such as **File** and **Edit** are ignored. The list is built in the current list for which you must define the following columns:

Column 1 (Character)	Column 2 (Character)
Menu instance name	Description for menu class (if one has been entered)

When you use the Clear list option you get column 1 only defined as #S5 with a 15 character column width. With this option, the command becomes reversible.

Menu instances from libraries other than the current library are prefixed with their library names. The flag is cleared if the command fails due to a shortage of memory.

Example

```
# Build a list of all menu instances installed on the
# main Omnis menu bar
Set current list lMenuList
Define list {lMenuName,lMenuDesc}
Build installed menu list

# Alternatively, you can use $makelist
Do $imenus.$makelist($ref.$name) Returns lMenuList
Do lMenuList.$redefine(lMenuName)
```

Build list columns list

Command group	Flag affected	Reversible	Execute on client
Lists	YES	YES	NO

Syntax

Build list columns list list-or-row-name ([Clear list])

Options

Clear list	If specified, the command empties the current list and defines it to have a single hash variable column before executing

Deprecated Command

This command has been deprecated and is no longer visible in the Code Assistant in the Code Editor (it will not appear when you type the first few characters), although it is still present in Omnis Studio and will continue to function if used in legacy code. You can show this command by disabling the appropriate Command Filter in the **Modify** menu in the Code Editor.

Description

This command builds a list containing the column names and data types of the current or specified list. This information is placed in the current list. If the current list contains one column, it contains the column names only. The current list column headings are ignored, but to obtain all the available information, you define the list with two columns as follows:

Col 1 (Character)	Col 2 (Character)
List Column name	List Column data type

The *Clear list* option clears and defines the current list to contain one column, #S5, so the column data types are not returned. With this option, the command becomes reversible.

The flag is cleared if the value of LIST.*$linemax* prevents a complete list from being built. The following method and the list of data it loads into the list illustrate the typical values produced:

Example

```
Do iMyList.$define(iPODate,iPONumber,iPOBatched,iSUContact,iITUnitPrice)
Set current list iColsList
Define list {iColName,iColType}
Build list columns list iMyList
# This provides the following values for iColsList
# iPODate - Short date 2000..2099
# iPONumber - Short integer (0 to 255)
# iPOBatched - Boolean
# iSUContact - Character 30
# iITUnitPrice - Number 2 dp
# Or you do the following:
Calculate iColsList as iMyList.$cols.$makelist($ref.$name,$ref.$coltype)
```

Build list from file

Command group	Flag affected	Reversible	Execute on client
Lists	YES	NO	NO

Syntax

Build list from file on field-name ([Exact match][,Use search][,Use sort])

Options

Exact match	If specified, the index value of the field in suitable records must equal the current value
Use search	If specified, the command uses the current search to select data
Use sort	If specified, the command uses the current sort field(s) to order the data

Deprecated Command

This command has been deprecated and is no longer visible in the Code Assistant in the Code Editor (it will not appear when you type the first few characters), although it is still present in Omnis Studio and will continue to function if used in legacy code. You can show this command by disabling the appropriate Command Filter in the **Modify** menu in the Code Editor.

Description

This command builds a list of data from the main file using a specified index field. The records are selected and corresponding field values added to the list in the order of the specified index field. You must set the main file before using the command.

If the Exact match option is specified, only records matching the current value of the specified field are added to the list. Similarly, if the Use search check box is selected, only records matching the current search class are added. In both cases, an error occurs if neither a field nor a search class is specified.

When large files are involved, that is, those that may require more than the maximum number of available lines (the value of LIST.*$linemax*), you can use the flag false condition to detect when an incomplete list is built.

Building a list using this command does not affect the current record buffer and does not clear 'Prepare for update' mode.

The Use sort option lets you use the database records in sorted order without first having to load them into a list. You use Set sort field to specify a sort field after which **Build list from file** (Use sort) creates a sorted table of records in memory before loading them into the list. The main advantage of this method is that the sort fields do not have to be read into the list at all. The Sort field order overrides the index field order but if the sort field is non-indexed, the index is used as the order in which to gather up records before sorting. Multi-level sorts are possible by using repeated Set sort field commands to accumulate the required sorting order. Since sort levels are cumulative you should first clear any existing ones with Clear sort fields.

Example

```
# This example compiles a list of all records sorted in order of descending fCustomers.Surname
# and within each value, in increasing fCustomers.FirstName order
Set current list iMyList
Set main file {fCustomers}
Define list {fCustomers.Surname,fCustomers.FirstName}
Clear sort fields
Set sort field fCustomers.Surname (Descending)
Set sort field fCustomers.FirstName
# Note fCustomers.CustomerID is not in the list
Build list from file on fCustomers.CustomerID (Use sort)
```

Build menu list

Command group	Flag affected	Reversible	Execute on client
Menus	YES	YES	NO

Syntax

Build menu list ([*Clear list*])

Options

Clear list	If specified, the command empties the current list and defines it to have a single hash variable column before executing

Description

This command builds a list containing the name of each menu class in the current library. The list is built in the current list for which the columns must have been defined. The columns are

Column 1 (Character)	Column 2 (Character)
Menu class name	Description for menu (if one has been entered)

The Clear list option clears the current list and redefines it to include only the #S5 field. With this option, the command becomes reversible but you get column 1 only.

Example

```
# Build a list of all menu classes in the current library
Set current list lMenuList
Define list {lMenuName,lMenuDesc}
Build menu list

# Alternatively, you can use $makelist
Do $menus.$makelist($ref.$name) Returns lMenuList
Do lMenuList.$redefine(lMenuName)
```

Build open window list

Command group	Flag affected	Reversible	Execute on client
Windows	YES	YES	NO

Syntax

Build open window list ([*Clear list*])

Options

Clear list	If specified, the command empties the current list and defines it to have a single hash variable column before executing

Description

This command builds a list containing the name of each window instance, starting with the topmost window instance. The window instance names are stored in the first column of the list. You can also return the position and size coordinates of each window instance in the second to fifth columns. The list is built in the current list for which you must define the following columns:

Col 1 (Character)	Col 2 (Long Int)	Col 3 (Long Int)	Col 4 (Long Int)
Window instance name	/left window coord	/top window coord	/right window coord

If you use the *Clear list* option, the list will contain one column only defined as #S5, so the window coordinates are not returned. Also, with the *Clear list* option selected, the command is reversible, that is, the list definition and contents are restored when the method terminates.

Example

```
# Build a list of open windows
Set current list lWindowList
Do lWindowList.$define(lName,lLeft,lTop,lRight,lBottom)
Build open window list

# Alternatively, notation can be used to build a list
# of open windows
Do $iwindows.$makelist($ref.$name) Returns lWindowList
Do lWindowList.$redefine(lName)
```

Build report list

Command group	Flag affected	Reversible
Reports and Printing	YES	YES

Syntax

Build report list ([Clear list])

Options

Clear list	If specified, the command empties the current list and defines it to have a single hash variable column before executing

Description

This command builds a list containing the name of each report class in the current library. The list is built in the current list for which the columns must have been defined. The columns are

Column 1 (Character)	Column 2 (Character
Report class name	Description for report (if one has been entered)

You get column 1 only when you use the Clear list option.

The *Clear list* option clears the current list and redefines it to include only the #S5 field. With this option the command becomes reversible.

Example

```
# Build a list of report classes in the current library
Set current list lReportList
Define list {lClass,lDesc}
Build report list

# Alternatively, you can use notation to build a list
# of report classes
Do $clib.$reports.$makelist($ref.$name,$ref.$desc) Returns lReportList
Do lReportList.$redefine(lClass,lDesc)
```

Build search list

Command group	Flag affected	Reversible	Execute on client
Searches	YES	YES	NO

Syntax

Build search list ([*Clear list*])

Options

Clear list	If specified, the command empties the current list and defines it to have a single hash variable column before executing

Description

This command builds a list containing the name of each search class in the current library. The list is built in the current list for which the columns must have been defined. The columns are

Column 1 (Character)	Column 2 (Character)
Search class name	Description for search (if one has been entered)

You get column 1 only when you use the Clear list option.

The *Clear list* option clears the current list and redefines it to include only the #S5 field. With the *Clear list* option, the command is reversible. The flag is cleared if the value of LIST.*$linemax* prevents a complete list from being built.

Example

```
# build a list of the available search classes
Set current list lSearchList
Build search list (Clear list )

# or use the following notation
Do $clib.$searches.$makelist($ref.$name) Returns lSearchList
```

Build window list

Command group	Flag affected	Reversible	Execute on client
Windows	YES	YES	NO

Syntax

Build window list ([Clear list])

Options

Clear list	If specified, the command empties the current list and defines it to have a single hash variable column before executing

Description

This command builds a list containing the name of each window class in the current library. The list is built in the current list for which you must define the following columns

Column 1 (Character)	Column 2 (Character)
Window class name	Description for window (if one has been entered)

You get column 1 only when you use the Clear list option, but the command becomes reversible.

The Clear list option clears the current list and redefines it to include only the #S5 field. With the Clear list option, the command becomes reversible.

Example

```
# Build a list of all window classes in the current library
Set current list lWindowList
Do lWindowList.$define(lName,lDesc)
Build window list

# Alternatively, notation can be used to build the list
# of window classes
Do $clib.$windows.$makelist($ref.$name,$ref.$desc) Returns lWindowList
Do lWindowList.$redefine(lName,lDesc)
```

Calculate

Command group	Flag affected	Reversible	Execute on c
Calculations	NO	YES	YES

Syntax

Calculate field-name as calculation

Description

This command assigns a new value to a data field or variable. The form of the command is “Calculate X as Y”, where X is a valid data field or variable name and Y is either a valid data field or variable name, value, calculation, or notation. When Calculate is executed the state of the flag is unchanged, unless #F is recalculated by this command.

You can use Calculate in a reversible block. The data field returns to its initial value when the method containing the block of reversible commands finishes.

Warning the Calculate command does not redraw a calculated field so if your field is on a window you must use the Redraw command or the $redraw() method after the Calculate command to reflect the change.

Operator Precedence

Mathematical expressions are evaluated using the operator precedence so that in the absence of brackets, * and / operations are evaluated before + and -. The full ordering from highest to lowest precedence is:

unary minus

* and /

- and -

>, <, >=, <=, <>, =

& and |

For example, if you execute the command “Calculate lVar1 as 10-2*3” the calculation part is evaluated as 10-(2*3)

Example

```
# set the local variable lVar1 equal to the contents of lVar2
Calculate lVar1 as lVar2

# set the local variable lPrice to 10.99 and lQty to 2
Calculate lPrice as 10.99
Calculate lQty as 2
```

```
# calculate the local variable lTotal as lPrice multiplied by lQty
Calculate lTotal as lPrice*lQty

# you can also operate on variables using notation, for example, calculate the
# local list variable lClassList as a list of all classes in the current library
Calculate lClassList as $clib.$classes.$makelist($ref.$name)

# however some operations are better performed using the Do command, for example
# bring the window instance wMywindow to the front
Do $iwindows.wMywindow.$bringtofront()
```

Call DLL

Command group	Flag affected	Reversible	Execute on client	P
External commands	YES	NO	NO	V

Syntax

Call DLL (library, procedure [,parameters...]) **Returns** return-value

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This command calls a procedure in a DLL, which you must have previously registered by calling Register DLL. The library is the name or pathname of the DLL containing the procedure specified by procedure; library and procedure must exactly match the values passed to Register DLL.

The parameters are passed to the procedure when it is called, and must match the type-definition passed to Register DLL. The return value of Call DLL is the return value of procedure, and has the type specified by the type-definition. Register and Call DLL commands support 64-bit type specifiers.

Example

```
# Flash the Omnis window to attract the user's attention
# Win32 API to get the main Omnis window: HWND GetActiveWindow(VOID)
Register DLL ('USER32.DLL','GetActiveWindow','J')
Call DLL ('USER32.DLL','GetActiveWindow') Returns lHWND
# Win32 API to Flash a window: BOOL FlashWindow(HWND, BOOL)
Register DLL ('USER32.DLL','FlashWindow','JJJ')
Call DLL ('USER32.DLL','FlashWindow',lHWND,1) Returns lResult
```

This example creates a file and loads the contents:

```
Register DLL ("KERNEL32.DLL","CreateFileA","JCJJJJJJ")
Register DLL ("KERNEL32.DLL","CloseHandle","JJ")
Register DLL ("KERNEL32.DLL","ReadFile","J,J,C32768,J,N,J")
Call DLL ("KERNEL32.DLL","CreateFileA","c:\MYBIGFILE.TXT",-1073741824,3,0,3,268435584,0) Returns #1
Call DLL ("KERNEL32.DLL","ReadFile",#1,#S1,32767,#49,0) Returns #50
Call DLL ("KERNEL32.DLL","CloseHandle",#1) Returns #50
Calculate #1 as binlength(#S1)
```

Call external routine

Command group	Flag affected	Reversible	Execute on client
Externals	NO	NO	NO

Syntax

Call external routine *routine-name* or *library-name/routine-name* (*parameters*) **Returns** *return-value*

Description

This command calls an external routine with mode ext_call and returns a value from the external in the specified return-field. The return value is placed in the specified field by the external code using the predefined field reference Ref_returnval with the functions SetFldVal or SetFldNval. The flag is set if the external routine is found and the call is made but this does not necessarily mean that the external code has executed correctly. The flag is cleared if the routine is not found. Note that the routine cannot use the flag to pass information back to the method.

You can pass parameters to the external code by enclosing a comma-separated list of fields and calculations. If you pass a field name, for example, Call external routine Maths1 (Num1,Num2), the external can directly alter the field value. Enclosing the field in brackets, for example, Call external routine Maths1 ((Num1),(Num2)), converts the field to a value and protects the field from alteration.

In the routine itself, the parameters are read using the usual GetFldVal or GetFldNval with the predefined references Ref_parm1, Ref_parm2, and so on, Ref_parmcnt gives the number of parameters passed. If the field name is passed as a parameter, you can use SetFldVal or SetFldNval with Ref_parm1, and so on, to change the field's value.

Example

```
Call external routine MathsLib/sqr (iNumber) Returns iNumber2
```

Cancel advises

Command group	Flag affected	Reversible	Ex
Exchanging data	NO	NO	NO

Syntax

Cancel advises *field-name* ([*All channels*])

Options

All channels	If specified, the command applies to all DDE channels, rather than just the current channel

Description

DDE command, Omnis as client. This command cancels one or more Request advises from the current channel. If you omit the field name, all Request advises to the current channel are canceled. If you specify a field name, all Request advises to the current channel which refer to that field name are canceled.

The command is addressed to the current channel only, and if the current channel is not open, an error occurs. No error occurs, however, if there are no Request advises commands to cancel.

If you use the All channels option, all channels are cancelled. There is no need to use a Cancel advises command before a Close DDE channel command.

When Omnis issues a Request advises to a DDE server, Omnis is in effect saying "Tell me if this value changes and send me an update". The Enter data command must be running to allow the incoming data to get through.

Example

```
Yes/No message {Do you want updates?}
If flag false
  Cancel advises (All channels)
  Quit method
Else
  Request advises iCompany {Company}
  Request advises iAddress {Address}
End If

Prepare for insert
Enter data
Update files if flag set
```

Cancel async method

Command group	Flag affected	Reversible	Execute on client
Methods	YES	NO	NO

Syntax

Cancel async method {*id-to-cancel* (*return-value-from-do-async-method*)}

Description

This command allows you to cancel the execution of a method that is executing as a result of a call to the Do async method command.

This command takes a single parameter id-to-cancel, which is the asynchronous call id returned by Do async method.

This command sets the flag if it has marked the async method for cancellation. Omnis only checks to see if the method is marked for cancellation after the completion of each method command, so cancellation may not occur immediately. Also, if you are executing a sensitive block of code, which should not be cancelled in this way, you can use the Begin critical block and End critical block commands around the sensitive code. Omnis will only cancel the method execution when the thread ends the critical block. If the flag is cleared, then either the asynchronous call id is invalid, or the method has finished. After successfully cancelling a method call, Omnis still sends the $asynccomplete message, but with an error text parameter that indicates that the call was cancelled.

Note

You can only call Cancel async method when running in the normal foreground thread.

Example

```
# iCallId was returned by Do async method
Cancel async method {iCallId}
```

Cancel prepare for update

Command group	Flag affected	Reversible	Execute c
Changing data	NO	NO	NO

Syntax

Cancel prepare for update

Description

This command cancels the Prepare for update mode and releases any semaphores which may have been set. You use the Prepare for edit/insert command to prepare Omnis for editing or insertion of records. It is usually followed by Update files which is the usual way of terminating the Prepare for... state but you can also terminate this state with Cancel prepare for update. It must be followed by commands which prevent an Update files command from being encountered.

When you execute a Prepare for... command in multi-user mode, semaphores are used to implement record locking. Cancel prepare for update neutralizes the effect of a Prepare for...command and releases all semaphores.

You can use this command within a timer method to implement a timed record release.

Example

```
Set timer method 600 sec TimerMethod
Prepare for edit
Enter data
Update files if flag set
Clear timer method
# TimerMethod
Yes/No message {Time's up, cancel edit?}
If flag true
  Cancel prepare for update
  Queue cancel
End If
```

Case

Command group	Flag affected	Reversible	Execute on client
Constructs	NO	NO	YES

Syntax

Case constant-value or expression

Description

The Case statement is part of a Switch construct that chooses one of an alternative set of options. The options in a Switch construct are defined by the subsequent Case commands. The Case command takes either a constant, field name, single calculation, or a comma-separated series of calculations. You must enclose string literals in quotes. Date values must match the date format in #FDT.

You can use the Break to end of switch command to jump out of the current Case statement and resume method execution after the *End Switch* command. Note you cannot use the *Break to end of loop* command to break out of a Switch construct.

Example

```
# Show the direction lPosition equals. eg. if lPosition equals 3 show 'South' in the ok message

Switch lPosition
  Case 1
    Calculate lDirection as 'North'
  Case 2
    Calculate lDirection as 'East'
  Case 3
    Calculate lDirection as 'South'
  Case 4
    Calculate lDirection as 'West'
End Switch
```

```
OK message {Position [lCount] = [lDirection]}

# Multiple conditions can be used in a comma-separated list to one Case statement.
# Default is used to specify commands that should run if the value is not one of
# those specified in the Case statements
Switch lDirection
  Case 'North','South'
    OK message {The direction is North or South}
  Case 'East','West'
    OK message {The direction is East or West}
  Default
    OK message {The direction is Unknown} ## # lDirection is none of the above
End Switch
```

CGIDecode

Command group	Flag affected	Reversible	Execute on client	P
External commands	YES	NO	NO	A

Syntax

CGIDecode (*stream*[,*mapplustospace* {Default kTrue}]) **Returns** *decoded-stream*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded,allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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.

Example

```
Calculate lStream as 'Name: Charlie Malone,Company: Omnis Software'
CGIEncode (lStream) Returns lEncodedStream
CGIDecode (lEncodedStream) Returns lDecodedStream
# lDecodedStream now contains the following:
# Name: Charlie Malone,Company: Omnis Software
Calculate lStream as 'Name: Charlie Malone+Friend,Company: Omnis Software'
CGIEncode (lStream) Returns lEncodedStream
CGIDecode (lEncodedStream,kFalse) Returns lDecodedStream
# lDecodedStream now contains the following:
# Name: Charlie Malone+Friend,Company: Omnis Software
CGIDecode (lEncodedStream) Returns lDecodedStream
# lDecodedStream now contains the following:
# Name: Charlie Malone Friend,Company: Omnis Software
# Note the + has been turned into a space character
```

CGIEncode

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

CGIEncode (*stream*[,*mapplustohex* {Default kFalse}]) **Returns** *encoded-stream*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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.

MapPlusToHex is an optional Boolean parameter which when true indicates that plus characters in the input stream are to be URL encoded as hex.

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.

Example

```
Calculate lStream as 'Name: Charlie Malone,Company: Omnis Software'
CGIEncode (lStream) Returns lEncodedStream
```

Change user password

Command group	Flag affected	Reversible	Execute on client
Libraries	NO	NO	NO

Syntax

Change user password

Description

This command opens the Password dialog in which the user can change the current password. The menus are redrawn and lists and variable values (apart from #UL) are unaffected.

If the current user is the master user, passwords in the #PASSWORDS class can be changed. In addition, the command gives the user the choice of using another password to re-enter the current library at a different user level, thus gaining access to different areas of the library. If a user re-enters at a different level, the value of #UL will change (within the range 0–8) to reflect that new user level.

Example

```
# Prompt the user for a password as specified in #PASSWORDS
# and display the current user level
Change user password
OK message {The current user level is [#UL]}
```

Change working directory

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

Change working directory (*path*) **Returns** *err-code*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This command changes the current directory in use under Windows or Linux. Wild cards are not allowed with this command.

On Windows, **Change working directory** only switches directories on the same drive, not between drives.

It returns an error code (See Error Codes), or zero if no error occurs.

Example

```
Change working directory ("c:\omnis\html") Returns lErrCode    ## windows
Change working directory ("/omnis/html") Returns lErrCode    ## linux
```

Check data

Command group	Flag affected	Reversible	Ex
Data management	YES	NO	NO

Syntax

Check data ([*Perform repairs*][,*Check data file structure*][,*Check records*][,*Check indexes*]) {*list-of-files* (F1,F2,..,Fn) (leave empty to select all)}

Options

Perform repairs	If selected,repairs to the data file are automatically carried out
Check data file structure	If specified,the command checks the overall structure of the data file
Check records	If specified,the command checks the records in the specified files
Check indexes	If specified,the command checks the indexes in the specified files

Description

This command checks the data for the specified file or list of files, and works only when one user is logged onto the data file. If you omit a file name or list of files, all the files with slots in the current data file are checked. If the specified file name does not include a data file name as part of the notation, the default data file for that file is assumed. If the file is closed or memory-only, the command does not execute and returns with the flag false.

There are Check data file structure, Check records, and Check indexes checkbox options. If none of these is specified, the command does nothing; if only Check data file structure is specified, the list of files is ignored. If Perform repairs is specified, any repairs required are automatically carried out, otherwise the results of the check are added to the check data log. The check data log is not opened by this command but is updated if already open.

If you are not running in single user mode, this command automatically checks that only one user is using the data file (the command fails with flag false if this is not true), and further users are prevented from logging onto the data until the command completes.

If a working message with a count is open while the command is executing, the count will be incremented at regular intervals. The command may take a long time to execute and it is not possible to cancel execution even if a working message with cancel box is open.

The command sets the flag if it completes successfully and clears the flag otherwise. It is not reversible.

Example

```
Check data (Check records) {fOrders}
If flag true
  Yes/No message {View Log?}
  If flag true
    Open check data log
  End If
Else
  OK message Error (Icon) {The check data file command could not be carried out//Please make sure that only on
End If
```

Check menu line

Command group	Flag affected	Reversible	Execute on client
Menus	NO	YES	NO

Syntax

Check menu line *line* or *instance-name/line*

Description

This command places a check mark on the specified line of a menu instance to show that the option has been selected. You specify the menu instance name and the number of the menu line you want to check.

You can remove the check mark with Uncheck menu line. If you use this command in a reversible block, the check mark is removed when the method terminates. Nothing happens if the menu instance is not installed on the menu bar.

Example

```
# Test whether a line in the menu instance is checked and
# either check or uncheck it accordingly.
Install menu mView
Test for menu line checked mView/Large
If flag true
  Uncheck menu line mView/Large
Else
  Check menu line mView/Large
End If
# Alternatively, you change the $checked property of a line
# in the menu instance using notation
Do $imenus.mView.$objs.Large.$checked.$assign(kTrue)
```

Clear all files

Command group	Flag affected	Reversible	Execute on client
Files	NO	YES	NO

Syntax

Clear all files

Description

This command clears the current record buffer of all file variables for all open libraries and all open data files, including any memory-only files. However, it does not clear the hash variables. Window instances are not automatically redrawn so you must follow it by Redraw if you want the screen to reflect the current state of the buffer.

This command is reversible for read-only and read-write files; the command reverses by re-reading each record into the current record buffer. Note that using this command in a reversible block with a memory-only file will clear the current record buffer for that file when the command reverses.

Example

```
# Clear all file variables from the current record buffer and
# redraw the current window instance
Clear all files
Do $cinst.$redraw()
```

Clear check data log

Command group	Flag affected	Reversible	Ex
Data management	NO	NO	NO

Syntax

Clear check data log

Description

This command clears the check data log, which stores all the results of a check data operation. To clear the log, there is no need for the log to be open.

Example

```
Check data (Check records) {fOrders}
If flag true
  Yes/No message {View Log?}
  If flag true
    Open check data log
    # after checking through the log...
    Yes/No message {Clear the log?}
    If flag true
      Clear check data log
    End If
  End If
Else
  OK message Error (Icon) {The check data file command could not be carried out//Please make sure that only on
End If
```

Clear class variables

Command group	Flag affected	Reve
Parameters and variables	NO	NO

Syntax

Clear class variables

Description

This command clears any class variables used within the class and clears the memory used for the class variables. Clear class variables is placed in a method within the class where you want to clear variables.

A class variable is initialized to empty or its initial value the first time it is referenced. It remains allocated until the class variables for its class are cleared. The class variables for all classes are cleared when the library file is closed.

Example

```
# Transfer values from class variables to instance
# variables and clear the class variables
Calculate cVar1 as 'my class Var1'
Calculate cVar2 as 'my class Var2'
Calculate cVar3 as 'my class Var3'
Calculate iVar1 as cVar1
Calculate iVar2 as cVar2
Calculate iVar3 as cVar3
Clear class variables ## all class variables are now empty
```

Clear data

Command group	Flag affected	Reversible	Execute on client
Clipboard	YES	NO	NO

Syntax

Clear data field-name ([Redraw field][,All windows])

Options

Redraw field	If specified, the command reloads affected window fields with the new value of the data field, after it has performed the operation; note that this takes the 'All windows' option into account
All windows	If specified, the command applies to all open window instances, rather than just the top open window instance

Description

This command clears the data from the specified field or current selection. The data is lost and is not placed on the clipboard. If you do not specify a field, the current field's data is cleared (assuming there is a selection).

In the case of a null selection when the cursor is merely flashing in a field and no characters are selected, **Clear data** will literally clear "nothing".

Example

```
# The following method is placed behind a entry field named 'Price' on a window and
# checks if the value entered is over 5000. If it is, the value entered into the field
# is cleared and the cursor remains in the field.
On evAfter
  If iPrice>5000
    Yes/No message {Is this price correct?}
    If flag false
      Clear data iPrice (Redraw field)
      Queue set current field {Price}
    End If
  End If
```

Clear DDE channel item names

Command group	Flag affected	Reversible	Ex
Exchanging data	NO	YES	NO

Syntax

Clear DDE channel item names

Description

DDE command, Omnis as client. This command clears all server data item names selected for use with a print-to-channel report. You use this command when exporting data via a DDE channel to another Windows application. The channel item names become the item names into which the server places the fields printed in the Omnis report.

Clear DDE channel item names clears all the item names set up with Set DDE channel item name.

Example

```
Set DDE channel number {2}
Open DDE channel {Excel|Sheet1}
Send to DDE channel
Set report name rMyReport
Clear DDE channel item names
Send command {[[TakeControl]} ;; double first [['s so Omnis accepts text
If flag true
  Set DDE channel item nam {R1,C1}
  Set DDE channel item nam {R2,C1}
  # ...
  Set DDE channel item name {R50,C1}
  Print report
End If
```

Clear find table

Command group	Flag affected	Reversible	Execute on
Finding data	NO	NO	NO

Syntax

Clear find table

Description

This command clears the find table for the current main file and releases the memory it used.

When a Find, Next or Previous command is encountered, Omnis uses the Index, Search and Sort field parameters to create a table of records (similar to a SQL Select table). This may simply be an existing index in which case no further processing takes place or, if there is a search and/or sort condition, a file may be scanned and a selection of records sorted in memory. If a Next or Previous returns an unexpected record or no record, this is probably because there is still a find table in existence from another Find operation.

For a large file, a substantial amount of RAM may be used.

Example

```
# Clear the find table after the first overdrawn account is found
Set main file {fAccounts}
Set search as calculation {fAccounts.Balance<0}
Find first on fAccounts.Code (Use search)
Clear find table
```

Clear line in list

Command group	Flag affected	Reversible	Execute on client
List lines	YES	NO	NO

Syntax

Clear line in list {*line-number* (*calculation*)}

Deprecated Command

This command has been deprecated and is no longer visible in the Code Assistant in the Code Editor (it will not appear when you type the first few characters), although it is still present in Omnis Studio and will continue to function if used in legacy code. You can show this command by disabling the appropriate Command Filter in the **Modify** menu in the Code Editor.

Description

This command clears the values stored in the specified line of the current list. You can specify the line number in a calculation, otherwise the current line (*LIST.$line*) is used. The flag is cleared if the list is empty or if the line is beyond the current end of the list.

Example

```
# Clear values from any lines in the list that have a
# balance equal to zero
Set current list lMyList
Define list {lName,lBalance}
Add line to list {('Fred',100)}
Add line to list {('George',0)}
Add line to list {('Harry',50)}
For each line in list from 1 to lMyList.$linecount step 1
  If lst(lBalance)=0
    Clear line in list
  End If
End For
# Alternatively you can use $clear to clear the values
# of a particular line
Do lMyList.1.$clear()
```

Clear list

Command group	Flag affected	Reversible	Execute on client
Lists	NO	YES	NO

Syntax

Clear list ([*Hash lists*])

Options

Hash lists	If specified, the command clears #L1-#L8 rather than the current list. When this option is specified, the command is

Deprecated Command

This command has been deprecated and is no longer visible in the Code Assistant in the Code Editor (it will not appear when you type the first few characters), although it is still present in Omnis Studio and will continue to function if used in legacy code. You can show this command by disabling the appropriate Command Filter in the **Modify** menu in the Code Editor.

Description

This command clears all the lines in the current list and frees the memory they occupy. It does not alter the definition of the list. If you use **Clear list** as part of a reversible block, the list lines will be reloaded when the method containing the reversible block finishes. The list is only reloaded if it occupies 50,000 bytes of storage or less. Executing Clear list for a smart list sets $smartlist to kFalse, meaning that it is no longer a smart list.

The All Lists option only clears the hash variable lists #L1 to #L8: all other lists including task, class, instance and local variable lists, are not cleared by this command.

The following method builds a list of data formats depending on the type of graph selected by the user. Before the method is built the list is cleared using the **Clear list** command; this ensures the list is initialized and completely empty of data.

Example

```
Set current list iMyList
Clear list
# or you can do it like this
Do iMyList.$clear()
```

Clear main & connected

Command group	Flag affected	Reversible	Execute on client
Files	NO	YES	NO

Syntax

Clear main & connected

Description

This command clears the memory of current records from the main file and any files connected to the main file. The windows are not automatically redrawn so you must follow it with a Redraw window-name command if you want the screen to reflect the current state of the buffer.

You can use **Clear main & connected** to release locked records to other users.

Example

```
# Clear the current record buffer of file variables from fAccounts
# and any connected file classes if insert is cancelled
# $construct of window class
Set main file {fAccounts}
Prepare for insert
Enter data
If flag false
  Clear main & con
End If
```

Clear main file

Command group	Flag affected	Reversible	Execute on client
Files	NO	YES	NO

Syntax

Clear main file

Description

This command clears the main file record from the current record buffer. The command does not clear the values taken from the other files.

The **Clear main file** command does not redraw the window so remember to include an explicit Redraw window command if you want the screen to reflect the contents of the buffer.

Example

```
# Clear the current record buffer of file variables from the main
# file fAccounts and redraw the current window instance
Set main file {fAccounts}
Clear main file
Do $cinst.$redraw()
```

Clear method stack

Command group	Flag affected	Reversible	Execute on client
Methods	NO	NO	NO

Syntax

Clear method stack

Description

This command cancels all currently executing methods and clears the method stack. A **Clear method stack** at the beginning of a method terminates all the methods in the chain which called the current method but without quitting the current method. $control() methods are not cleared.

As each method calls another, a return point is stored so that control can pass to the command following Do method or Do code method as the called method terminates. When the current method terminates, control returns to the method which was running before it was called.

The **Clear method stack** command clears all the return points and is used if the method commences a completely new operation. This command followed by a Quit method is the same as Quit all methods.

WARNING It is unwise to clear the method stack if local variables have been passed as fieldname parameters and you continue executing the current method. This will break all local variables on the stack.

Example

```
# Calling method
Calculate iMyVar as 1
Do method Message
# the following message never gets displayed
Do iMyVar+1
OK message {iMyVar=[iMyVar]}
# Method Message
Clear method stack
Do iMyVar+1
# This message prints iMyVar=2
OK message {iMyVAR=[iMyVar]}
Quit method
```

Clear range of fields

Command group	Flag affected	Reversible	Execute on client	P
Files	NO	YES	NO	A

Syntax

Clear range of fields first-data-name **to** final-data-name

Description

This command clears the specified range of fields from the current record buffer.

Note that *first-data-name* and *last-data-name* identify the first and last field of the range to be cleared, in the order that the fields occur in the current record buffer. In certain current record buffers, for example the instance variables of an instance, the order of the fields in the current record buffer is the order in which the fields were created, not the alphabetic order in which they are displayed in the variable pane of the method editor.

When used in a reversible block, the fields cleared are restored when the method terminates.

Example

```
# Clear the current record buffer of fields Surname to Balance
# from fAccounts and redraw the current window instance
Clear range of fields fAccounts.Surname to fAccounts.Balance
Do $cinst.$redraw()
```

Clear search class

Command group	Flag affected	Reversible	Execute on client
Searches	NO	YES	NO

Syntax

Clear search class

Description

This command clears the current search class so you can print a report using all records. This also frees the memory required by the search class.

If you use **Clear search class** in a reversible block, the search class reverts to its former setting when the method terminates.

Example

```
Set report name rMyReport
Set search name sMySearch
# sys(81) returns the current search class
Yes/No message Use Search (Icon) {Do you wish to use the search class '[sys(81)]' ?}
If flag false
  Clear search class
End If
Print report (Use search)
```

Clear selected files

Command group	Flag affected	Reversible	Execute on client
Files	NO	YES	NO

Syntax

Clear selected files {list-of-files (F1,F2,..,Fn)}

Description

This command clears the current record buffer of records from the specified files. The command is particularly useful in a multi-user system where it may be necessary to remove only certain files so that they are not locked.

In the method editor, a list of files is displayed. You can Ctrl/Cmnd-click on the file names to select multiple names. If no file name or file list is specified, the command does nothing.

This command is reversible for read-only and read-write files; the command reverses by re-reading each record into the current record buffer. Note that using this command in a reversible block with a memory-only file will clear the current record buffer for that file when the command reverses.

Example

```
# Clear the current record buffer of records from fAccounts
# and fInvoices and redraw the current window instance
Clear selected files {fAccounts,fInvoices}
Do $cinst.$redraw()
```

Clear sort fields

Command group	Flag affected	Reversible	Execute on clien
Sort fields	NO	YES	NO

Syntax

Clear sort fields

Description

This command removes the sort fields that are currently active. This enables the data to be printed without any sorting taking place. Alternatively, the command removes the current sort fields so you can specify new sort levels with *Set sort field*.

If you use **Clear sort fields** in a reversible block, the original sort values are restored when the method terminates.

Example

```
# Remove the current sort fields and then set the sort
# field as Surname
Clear sort fields
Set sort field fAccounts.Surname
Set report name rMyReport
Send to screen
Print report
```

Clear timer method

Command group	Flag affected	Reversible	Execute on client	P
Methods	NO	YES	NO	A

Syntax

Clear timer method

Description

This command clears or cancels the current timer method. Usually a timer method remains in operation until the library is closed or an error occurs. In a reversible block, the current timer method is restored when the method terminates.

Example

```
# Clear the timer method after it is called so that is
# only called once
Set timer method 5 sec Timer

# method Timer
OK message {Timer method triggered once only}
Clear timer method
```

Clear trace log

Command group	Flag affected	Reversible	Execute on client
Debugger	NO	NO	NO

Syntax

Clear trace log

Description

This command clears the trace log.

Close all designs

Command group	Flag affected	Reversible	Execute on client
Classes	YES	NO	NO

Syntax

Close all designs

Description

This command closes all the design windows currently open, including all instances of the method editor.

Example

```
Close all designs
```

Close all windows

Command group	Flag affected	Reversible	Execute on client
Windows	NO	NO	NO

Syntax

Close all windows

Description

This command closes all open window instances in all open libraries, and automatically cancels any working message. The **Close all windows** command does not close private instances which do not belong to the current task.

Example

```
# Prompt to close all open windows
Yes/No message {Do you wish to close all windows ?}
If flag true
  Close all windows
End If
# Alternatively, the $sendall command can be used to close
# all windows
Do $root.$iwindows.$sendall($ref.$close())
```

Close check data log

Command group	Flag affected	Reversible	Ex
Data management	NO	NO	NO

Syntax

Close check data log

Description

This command closes the check data log if it is open. The command is not reversible and the flag is not affected.

Example

```
Check data (Check records) {fOrders}
If flag true
  Yes/No message {View Log?}
  If flag true
    Open check data log (Do not wait for user)
  End If
  # leave log window open
Else
  OK message Error (Icon) {The check data file command could not be carried out//Please make sure that only on
End If
# now close log
Close check data log
```

Close data file

Command group	Flag affected	Reversible	Execute on client
Data files	YES	NO	NO

Syntax

Close data file {*internal-name* (leave empty to close all)}

Description

This command closes the open data file with the specified internal name, or closes all the open data files if no name is specified. It sets the flag if at least one data file is closed. It clears the flag and does nothing (that is, does not generate a runtime error) if the specified internal name does not correspond to an open data file.

Note that data files have a notation property $allowclose, which when set to kFalse, prevents **Close data file**, the data file notation, and the Data File Browser from closing the file.

Example

```
# check the $allowclose property of myDataFile
If $root.$datas.myDataFile.$allowclose
  Close data file {myDataFile}
End If
```

Close DDE channel

Command group	Flag affected	Reversible	Ex
Exchanging data	NO	NO	NO

Syntax

Close DDE channel ([*All channels*])

Options

All channels	If specified, the command applies to all DDE channels, rather than just the current channel

Description

DDE command, Omnis as client. This command closes the current channel. If you use the *All channels* option, all open DDE channels are closed. No error occurs if the current channel is not open.

Example

```
Set DDE channel number {2}
Open DDE channel {Omnis|Country}
If flag false
  OK message {The Country library is not running}
Else
  Do method TransferData
  Close DDE channel
  OK message {Update finished}
End If
```

Close design

Command group	Flag affected	Reversible	Execute on client
Classes	YES	NO	NO

Syntax

Close design {*class-name*}

Description

This command closes the specified design class. Trying to close a class which is not open simply clears the flag.

Example

```
Close design
```

Close file

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

Close file (*refnum*) **Returns** *err-code*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This command closes a file previously opened by the Open file command. You specify the file reference number returned by Open file in refnum. You should call **Close file** for each files you open with Open file, when you have finished using the file.

It returns an error code (See Error Codes), or zero if no error occurs.

Example

```
# read a text file then close it

Calculate lPathname as con(sys(115),'html',sys(9),'serverusagetask.htm')
Open file (lPathname,lRefNum) Returns lErrCode ## opens the file
Read file as character (lRefNum,lFile) Returns lErrCode ## reads the file contents into lFile
Close file (lRefNum) Returns lErrCode ## now close the file
```

Close import file

Command group	Flag affected	Reversi
Importing and Exporting	NO	NO

Syntax

Close import file

Description

This command closes the current import file. You should use it once the data has been read in.

Example

```
# Import from a csv file called myImport.txt in the root of your omnis tree
Calculate lImportPath as con(sys(115),'myImport.txt')
Set import file name {[lImportPath]}
Prepare for import from file {Delimited (commas)}
Import data lImportList
End import
Close import file
```

Close library

Command group	Flag affected	Reversible	Execute on client
Libraries	YES	NO	NO

Syntax

Close library *internal-name* (leave empty to close all)

Description

This command closes the open library file with the specified internal name, or closes all the open library files if no name is specified. It sets the flag if at least one library file is closed. It clears the flag and does nothing if the specified internal name does not correspond to an open library.

Note that the internal name for a library defaults to its physical file name from which the path and DOS extension has been removed. The Open library command also lets you specify the internal name (see the example below).

Closing a library closes all windows, reports, and menus belonging to that library which are open or installed. It also disposes of the CRBs for the file classes and class variables belonging to that library, closes all lookup files opened by that library, and if there is a running method from that library on the stack, clears the method stack. If the method stack is cleared, the command following the current executing command will not execute, and it is not possible to test the flag value returned from the command.

Example

```
# Open and close the library mylib.lbs from the root
# of your Omnis studio tree
Calculate lLibPath as con(sys(115),'mylib.lbs')
Open library (Do not close others) [lLibPath],MYLIB
If flag true
  Yes/No message {Close Library ?}
  If flag true
    Close library MYLIB
  End If
End If
```

Close lookup file

Command group	Flag affected	Reversible	Execute on client
Data files	YES	NO	NO

Syntax

Close lookup file {*lookup-name*}

Description

This command closes the lookup file which matches the reference name given in the parameters. Each lookup file is given a reference label when it is opened. In this example it is “City”.

If the reference label given in the Open lookup file command is omitted, you can omit the lookup name in the Close lookup file command. If the specified lookup file is closed, the flag is set; if the lookup file doesn’t exist, the flag is cleared.

Example

```
Open lookup file {City,Lookup.df1,fCities}
If flag true
  OK message {The city you require is [lookup('City','I',2)]}
End If
Close lookup file {City}
```

Close other windows

Command group	Flag affected	Reversible	Execute on client
Windows	NO	NO	NO

Syntax

Close other windows

Description

This command closes all but the top window instance. As window instances are not automatically closed in Omnis, you can use this command to close all window instances except the top window instance. The **Close other windows** command does not close private instances which do not belong to the current task.

Example

```
# Close all other windows
If len(sys(51)) ## more than 1 window open
  Close other windows
End If
```

Close port

Command group	Flag affected	Reversible
Report destinations	NO	NO

Syntax

Close port

Description

This command closes the current port. You should use it after the data has been transferred.

Example

```
Set port name {COM1:}
Set port parameters {1200,n,7,2}
Prepare for import from port {One field per line}
Repeat
  Import field from file int lImportField
Until lImportField='start data'
Do method ImportData
Close import file
```

Close print or export file

Command group	Flag affected	Reversible
Report destinations	NO	NO

Syntax

Close print or export file

Description

This command closes the current print or export file. You use it after the data has been written to the file. If the file is left open, subsequent data printed to the file is added to the end of the earlier data.

Example

```
Send to file
Calculate lPrintFileName as con(sys(115),'myPrintedReport.txt')
Set print or export file name {[lPrintFileName]}
Set report name rMyReport
Print report
Close print or export file
```

Close task instance

Command group	Flag affected	Reversible	Execute on client
Tasks	NO	NO	NO

Syntax

Close task instance *instance-name*

Description

This command closes the specified task instance.

Example

```
Close task instance tkMyTask
# or do it like this
Do $itasks.tkMyTask.$close()
```

Close top window

Command group	Flag affected	Reversible	Execute on client
Windows	YES	NO	NO

Syntax

Close top window

Description

This command closes the top window instance. As window instances are not automatically closed in Omnis, you can use this command to close the top window. No error occurs if there is no window open. This command clears the flag and does nothing if the top window is a private instance not belonging to the current task.

Example

```
# Close the top window if it is called 'wMyWindow'
If sys(50)='wMyWindow'
  Close top window
End If
# Alternatively, use notation to close the top window
Do $topwind.$close()
```

Close trace log

Command group	Flag affected	Reversible	Execute on client
Debugger	NO	NO	NO

Syntax

Close trace log

Description

This command closes the trace log.

Close window instance

Command group	Flag affected	Reversible	Execute on client
Windows	YES	NO	NO

Syntax

Close window instance *window-instance-name*

Description

This command closes the specified window instance. **Close window instance** clears the flag and does nothing if the window is a private instance belonging to the current task. Alternatively you can use the $close() method to close a window instance.

Example

```
Test for window open {wMyWindow}
If flag true
  Close window instance wMyWindow
End If
# Alternatively, you can do it like this
Do $root.$iwindows.wMyWindow.$close()
```

Close working message

Command group	Flag affected	Reversible	Execute c
Message boxes	NO	NO	NO

Syntax

Close working message

Description

This command closes the current working message. No error occurs if there is no working message displayed. Working messages close themselves when methods stop running and control returns to the user. Once a working message is displayed, a call to another method leaves the message on the window. The message is not cleared automatically until the first method ends.

Example

```
# Close the working message before this method
# has finished
Working message {Processing Record [lCount]}
For lCount from 1 to 20000 step 1
  Redraw working message
End For
Close working message
For lCount from 1 to 50000 step 1
  Calculate lValue as lValue+lCount
End For
```

Context help

Command group	Flag affected	Reversible
Operating system	YES	NO

Syntax

Context help {*command* (*parameters*)}

Description

This command provides context help to the user: note this only applies to fat client or desktop apps, not web & mobile apps created using the JavaScript Client.

You specify a command mode option, and depending on the mode you can specify the help file name and context id. The command mode options are constants listed in the Catalog.

kHelpContextMode
initiates context help mode, showing a '?' cursor.

kHelpContext ('helpfile name', context id)
opens a general help window for the topic specified.

kHelpContextPopup ('helpfile name', context id)
opens a popup help window for the topic specified.

kHelpContents ('helpfile name')
opens the help file at the contents page.

kHelpQuit ('helpfile name')
closes window mode help.

Some options do not work on all platforms.

To implement context help for an object or area, you set the help id as a decimal value in the $helpid property of a class or object, including windows, menus, and toolbars. You can make your custom help file which must be placed in the Help folder and the name entered in the library preference property $clib.$prefs.$helpfilename.

When the user clicks on an object with the help cursor or presses the F1/Help key, Omnis looks for the help id. If it finds none for a window object, menu line, or toolbar control, it then looks in the next higher containing object.

Example

```
# Show the file index.htm from the omnis help folder
# in the standard help window
Context help {kHelpContext ('omnis','index')}
# Show ? cursor and awaits click, when user clicks, shows a popup
# window with topic $cobj.$helpid from $clib.$prefs.$helpfilename
# located in the Help folder
Context help {kHelpContextMode}
```

Copy file

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

Copy file (*from-path* [,*to-path*]) **Returns** *err-code*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This command makes a copy of the file specified in from-path. The to-path is the path to destination folder into which the file will be copied; the file to be copied must not already exist in the destination folder. If you omit to-path, a copy of the file named in from-path is created in the current directory using the same name with the extension ".BAK".

When constructing the path to a file or folder, you can use sys(9) to insert the correct path delimiter for the current platform: \ (back-slash) on Windows, or / (forward-slash) for Unix and 64-bit macOS (: colon on 32-bit macOS). In addition, you can use sys(115) to return the full pathname of the folder containing the Omnis executable, including the terminating path separator, which might be useful to reference files in the Omnis tree.

It returns an error code (See Error Codes), or zero if no error occurs.

Example

```
Calculate lPathname as con(sys(115),'html',sys(9),'serverusagetask.htm')
Calculate lNewPath as con(sys(115),'html',sys(9),'serverusagetask2.htm')
Copy file (lPathname,lNewPath) Returns lErrCode
# copies the file in lPathName to the filename contained in lNewPath
```

Copy list definition

Command group	Flag affected	Reversible	Execute on client
Lists	YES	NO	NO

Syntax

Copy list definition list-or-row-name ([Clear list])

Options

Clear list	If specified, the command empties the current list and removes its column definitions before executing

Deprecated Command

This command has been deprecated and is no longer visible in the Code Assistant in the Code Editor (it will not appear when you type the first few characters), although it is still present in Omnis Studio and will continue to function if used in legacy code. You can show this command by disabling the appropriate Command Filter in the **Modify** menu in the Code Editor.

Description

This command redefines the column headings of the current list by copying the columns and data structure from the specified list. If the current list contains data and you do not clear the list, no change is made to the internal structure of the list; in this case, columns are neither added nor removed, merely renamed and the command is similar to Redefine list.

When the current list is empty or the Clear list option chosen, the command is the equivalent to 'Define the list so that it matches the specified list'.

Example

```
Set current list iList1
Define list {iCol1Date,iCol2Num,iCol3Char}
Add line to list
Set current list iList2
Define list {iCol4Date,iCol5Num,iCol6Char}
```

```
Add line to list
# now change the definition of iList2 to match iList1
Copy list definition iList1 (Clear list )
# or you can do it like this
Do iList2.$copydefinition(iList1)
```

Copy to clipboard

Command group	Flag affected	Reversible	Execute on client
Clipboard	YES	NO	NO

Syntax

Copy to clipboard *field-name*

Description

This command copies the contents of the specified field or current selection and places it on the clipboard. In the case of a null selection when the cursor is merely flashing in a field and no characters are selected, the **Copy to clipboard** command will literally copy "nothing".

Example

```
# Copy one field to another then clear the first field
Copy to clipboard iName
Paste from clipboard iDeliveryName (Redraw field)
Clear data iName (Redraw field)
```

Create data file

Command group	Flag affected	Reversible	Execute on client
Data files	YES	NO	NO

Syntax

Create data file ([Do not close other data]) {file-name, internal-name}

Options

Do not close other data	If specified, the command does not close all open data files before opening the specified data

Description

This command creates and opens a new and empty, single segment data file, which becomes the "current" data file. You can specify the path name of the file to be created and the internal name for the open data file.

The *Do not close other data* option lets you have multiple open data files. If you uncheck this option ,all open data files are closed even if the command fails.

If the disk file with the specified path name cannot be created (and opened), the flag is cleared. Otherwise, the flag is set if the data file is successfully created and opened.

WARNING: If the file and path name is the same as an existing data file, all segments for that data file are deleted before the new file is created. If the data file was open, it is closed and deleted; a new and empty data file is then reopened.

Example

```
Yes/No message {Do you wish to add a new company?}
If flag true
  # method to do some preparatory code for the new datafile and generate the company name
  Do method Insert Company
  # creates a datafile in the same folder as the omnis executable
  # the name of the datafile is the value of the character variable iCompany
  Create data file (Do not close other dat) {(con(sys(115),iCompany,'.df1')/[iCompany]}
End If
# or do it like this
Do $datas.$add(con(sys(115),iCompany,'.df1'),kTrue,[iCompany])
```

Create directory

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

Create directory (*path*) **Returns** *err-code*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This command creates the directory named in path. The directory must not already exist. **Create directory** does not create intermediate directories. It only creates the last directory name in path.

When constructing the path to a file or folder, you can use sys(9) to insert the correct path delimiter for the current platform: \ (backslash) on Windows, or / (forward-slash) for Unix and 64-bit macOS (: colon on 32-bit macOS). In addition, you can use sys(115) to return the full pathname of the folder containing the Omnis executable, including the terminating path separator, which might be useful to reference files in the Omnis tree.

It returns an error code (See Error Codes), or zero if no error occurs.

Example

```
Calculate lDirName as con(sys(115),'MyNewDirectory')
# create the new directory in the root of your omnis tree
Create directory (lDirName) Returns lErrCode
```

Create file

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

Create file (*path*) **Returns** *err-code*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This command creates the file specified in path. Every directory or folder in path must already exist. **Create file** does not create directories or folders.

When constructing the path to a file or folder, you can use sys(9) to insert the correct path delimiter for the current platform: \ (back-slash) on Windows, or / (forward-slash) for Unix and 64-bit macOS (: colon on 32-bit macOS). In addition, you can use sys(115) to return the full pathname of the folder containing the Omnis executable, including the terminating path separator, which might be useful to reference files in the Omnis tree.

It returns an error code (See Error Codes), or zero if no error occurs.

Example

```
Calculate lPathname as con(sys(115),'MyNewFile.txt')
# create the new file in the root of your omnis tree
Create file (lPathname) Returns lErrCode
```

Create library

Command group	Flag affected	Reversible	Execute on client
Libraries	YES	NO	NO

Syntax

Create library ([Do not close others]) library-pathname, internal-name

Options

Do not close others	If specified, the command does not close all open libraries before opening the specified library

Description

This command creates and opens a new library file. You specify the full pathname and internal name of the library. The internal name is an alias that you supply and use in your methods to refer to that library file.

If no internal name is specified, the default internal name is the disk name of the file with the path name and suffix removed. For example, under Windows the internal name for 'c:\myfiles\mylib.lbs' is MYLIB. Similarly, under macOS the internal name for '/my-files/mylib.lbs' is 'mylib'.

A **Do not close others** option can also be specified so that you can open multiple libraries. If the disk file with the specified pathname cannot be created (and opened), the flag is cleared and no libraries are closed. Otherwise, if the option is not specified, all other open libraries are closed (see Close library for the consequences of closing a library).

WARNING If the path name is the same as an existing library, the existing library is overwritten. If the existing library is open, it is closed and deleted and a new, empty library is opened.

Example

```
# Create a library named mylib.lbs in the root of your Omnis Studio tree
Calculate lLibPath as con(sys(115),'mylib.lbs')
Create library (Do not close others) [lLibPath]
If flag true
  OK message {Library created!}
End If
```

Cut to clipboard

Command group	Flag affected	Reversible	Execute on client
Clipboard	YES	NO	NO

Syntax

Cut to clipboard field-name ([Redraw field][,All windows])

Options

Redraw field	If specified, the command reloads affected window fields with the new value of the data field,after it has performed that this takes the 'All windows' option into account
All windows	If specified, the command applies to all open window instances, rather than just the top open window instance

Description

This command cuts the contents of the specified field or current selection and places it on the clipboard. In the case of a null selection when the cursor is merely flashing in a field and no characters are selected, **Cut to clipboard** will literally cut "nothing".

Example

```
# Cut iName to the clipboard and paste it into iDeliveryName
Cut to clipboard iName (Redraw field)
Paste from clipboard iDeliveryName (Redraw field)
```

Default

Command group	Flag affected	Reversible	Execute on clien
Constructs	NO	NO	YES

Syntax

Default

Description

This command marks the block of commands to be run when there is no matching case in a Switch statement. When a Switch–Case construct is used, the Default command marks the start of a block of commands that are executed if none of the preceding Case statements are executed.

Example

```
# Sound the bell if lName is not equal to Fred or Jim
Switch lName
  Case 'Fred'
    OK message {Fred}
  Case 'Jim'
    OK message {Jim}
  Default
    OK message (Sound bell ) {Neither Fred nor Jim}
End Switch
```

Define list

Command group	Flag affected	Reversible	Execute on client
Lists	NO	YES	NO

Syntax

Define list {*list-of-field-or-file-names* (F1,F2..F3,F4)}

Deprecated Command

This command has been deprecated and is no longer visible in the Code Assistant in the Code Editor (it will not appear when you type the first few characters), although it is still present in Omnis Studio and will continue to function if used in legacy code. You can show this command by disabling the appropriate Command Filter in the **Modify** menu in the Code Editor.

Description

This command defines the variables or file class field names to be used as the column definitions for the current list; it should follow Set current list. The variables or fields used in the definition also describe the data type and length for each column of data held. This command clears the definition and data in the current list. When reversed, the contents and definition of the current list are restored to their former values. Duplicate names are ignored in your list of variables or fields.

Example

```
Set current list iList1
# define columns iCol1Date, iCol2Num & iCol3Char for the current list
Define list {iCol1Date,iCol2Num,iCol3Char}
# same as before but ignores the duplicate reference to iCol3Char
Define list {iCol1Date,iCol2Num,iCol3Char,iCol3Char}
# define the list based upon all the columns in the file class fCustomers
Define list {fCustomers}
# Alternatively, you can avoid using Set Current List by using the following notation
Do iList1.$define(iCol1Date,iCol2Num,iCol3Char)
# define the list based upon a table,schema or query class
Do iList1.$definefromsqlclass('myTableOrSchemaOrQueryClass')
# FIXED LENGTH COLUMNS
# Normally, the length of a column is set by the type or length of the variable or field defined for
# the column, therefore the column length for a default character variable would be 10 million.
# However, when you define the list you can truncate the data stored in the column using
# VariableName/N. For example to use only the first 10 characters of the variable iCol3Char in column 3
Define list {iCol1Date,iCol2Num,iCol3Char/10}
```

Define list from SQL class

Command group	Flag affected	Reversible	Execute on client
Lists	NO	YES	NO

Syntax

Define list from SQL class *query*, *schema*, or *table-name*(*parameters*)

Deprecated Command

This command has been deprecated and is no longer visible in the Code Assistant in the Code Editor (it will not appear when you type the first few characters), although it is still present in Omnis Studio and will continue to function if used in legacy code. You can show this command by disabling the appropriate Command Filter in the **Modify** menu in the Code Editor.

Description

This command defines the column names and data types for the current list based on the specified schema, query, or table class.

This results in the creation of a new table instance associated with the list. If the sql-class-name refers to a table class, the command passes the parameters to the $construct method of the table class.

When reversed, the contents and definition of the list are restored to their former values.

Example

```
Set current list iMyList
Define list from SQL class sMySchema
# or do it this way
Do iMyList.$definefromsqlclass('sMySchema')
```

Delete

Command group	Flag affected	Reversible	Execute c
Changing data	YES	NO	NO

Syntax

Delete

Description

This command deletes the current record in the main file without prompting the user to confirm the command, so you should use it with caution. The flag is set if the record is deleted, or cleared if there is no main file record. The flag is also cleared if the Do not wait for semaphores option is on and the record is locked.

Example

```
# The following example deletes records selected by a search class.
Set main file {fAccounts}
Set search name sOverDrawn
Find first on fAccounts.Code (Use search)
Repeat
  Delete
  Next
Until flag false
# This example checks the semaphore and tells the user if the record is locked:
Do not wait for semaphores
Delete
If flag false
  OK message (Sound bell ) {Record in use and can't be deleted}
End If
```

Delete class

Command group	Flag affected	Reversible	Execute on client
Classes	YES	NO	NO

Syntax

Delete class {*class-name*}

Description

This command deletes the specified library class. It is not possible to delete a file class, an installed menu or an open window. It is also not possible to delete a class if one of its methods is currently executing, that is, if it is somewhere on the method stack. Deleting a class does not reduce the library file size. It does, however, create free library file blocks so that creation of another class may be possible without further increase in library size. Errors, such as attempting to delete a name that does not exist, simply clear the flag and display an error message.

Example

```
Delete class {sUser}
```

Delete data

Command group	Flag affected	Reversible	Ex
Data management	YES	NO	NO

Syntax

Delete data {*file-name*}

Description

This command deletes all the data and indexes for a specified file in a data file. The data and indexes for a file class are called a “slot”. You can delete a slot only if and when one user is logged onto the data file.

If a specified file name does not include a data file name as part of the notation, the default data file for that file is assumed. If the file is closed or memory-only, the command does not execute and returns flag false. If you are not running in single user mode, the command automatically tests that only one user is using the data file (the command fails with the flag false if this is not true), and further users are prevented from logging onto the data until the command completes.

If a working message with a count is open while the command is executing, the count will be incremented at regular intervals. The command may take a long time to execute, and it is not possible to cancel execution even if a working message with cancel box is open. The command sets the flag if it completes successfully and clears the flag otherwise. It is not reversible.

Example

```
Delete data {fCustomers}
If flag true
  OK message {Data for fCustomers has been deleted}
Else
  OK message Error {Data could not be deleted}
End If
```

Delete file

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

Delete file (*path*) **Returns** *err-code*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This command permanently deletes the file specified by path.

It returns an error code (See Error Codes), or zero if no error occurs.

When constructing the path to a file or folder, you can use sys(9) to insert the correct path delimiter for the current platform: \ (back-slash) on Windows, or / (forward-slash) for Unix and 64-bit macOS (: colon on 32-bit macOS). In addition, you can use sys(115) to return the full pathname of the folder containing the Omnis executable, including the terminating path separator, which might be useful to reference files in the Omnis tree.

Example

```
Calculate lPathname as con(sys(115),'html',sys(9),'serverusagetask.htm')
Calculate lNewPath as con(sys(115),'html',sys(9),'serverusagetask2.htm')
Copy file (lPathname,lNewPath) Returns lErrCode ## copies the file in lPathName to the filename contained in l
Does file exist (lNewPath) Returns lStatus ## see if the file exists
If lStatus
  Delete file (lNewPath) Returns lErrCode ## delete it
End If
```

Delete line in list

Command group	Flag affected	Reversible	Execute on client
List lines	YES	NO	NO

Syntax

Delete line in list {*line-number* (*calculation*)}

Deprecated Command

This command has been deprecated and is no longer visible in the Code Assistant in the Code Editor (it will not appear when you type the first few characters), although it is still present in Omnis Studio and will continue to function if used in legacy code. You can show this command by disabling the appropriate Command Filter in the **Modify** menu in the Code Editor.

Description

This command deletes the specified line of the current list by moving all the lines below the specified line up one line. If the line number is not specified or if it evaluates to 0, the current line *LIST.$line* is deleted. The line in a list selected by the user can determine the value of LIST.$line and is the line deleted if no parameters are specified. LIST.$line is unchanged by the command unless it was the final line and that line is deleted; in this case LIST.$line is set to the new final line number. The command never releases any of the memory used by the list.

The flag is cleared if the list is empty or if the line is beyond the current end of the list; otherwise, the flag is set.

Example

```
# Delete all but the first 2 lines in the list
Set current list lMyList
Define list {lName,lAge}
Add line to list {('Fred',10)}
Add line to list {('George',20)}
Add line to list {('Harry',22)}
Add line to list {('William',31)}
Add line to list {('David',62)}
```

```
While lMyList.$linecount>2
  Delete line in list {1}
End While
# Alternatively you can use $remove to delete a line from a list
Do lMyList.$remove(1)
```

Delete selected lines

Command group	Flag affected	Reversible	Execute on client
List lines	YES	NO	NO

Syntax

Delete selected lines

Deprecated Command

This command has been deprecated and is no longer visible in the Code Assistant in the Code Editor (it will not appear when you type the first few characters), although it is still present in Omnis Studio and will continue to function if used in legacy code. You can show this command by disabling the appropriate Command Filter in the **Modify** menu in the Code Editor.

Description

This command deletes all the selected lines from the current list. This is carried out in memory and has no effect on the lists stored in the data file unless a Prepare for Insert/ Edit command is performed.LIST.$line is unaffected unless it is left at a value beyond the end of the list, in which case it is set to LIST.$linecount.

Example

```
# Build a list and delete all lines except line 3
Set current list lMyList
Define list {lCol1}
For lCount from 1 to 10 step 1
  Add line to list {(lCount)}
End For
Select list line(s) (All lines)
Invert selection for line(s) {3}
Delete selected lines
```

Delete with confirmation

Command group	Flag affected	Reversible	Execute on client	P
Changing data	YES	NO	NO	A

Syntax

Delete with confirmation {*message*}

Description

This command displays a message asking the user to confirm or cancel the deletion and, if confirmation is granted, deletes the current record in the main file. An error is reported if there is no main file.

If a message is not specified, Omnis uses a default message. The message can contain square-bracket notation which is evaluated when the command is executed. If the current record is deleted, the flag is set, otherwise it is cleared. If the Do not wait for semaphores option is on, the flag is cleared if the record is locked.

Example

```
# This example allows selected records in the main file to be deleted:
Set main file {fAccounts}
Set search as calculation {fAccounts.Balance<0}
Find first on fAccounts.Code (Use search)
While flag true
  Delete with confirmation {Delete [fAccounts.Surname]'s record?}
  Next (Use search)
End While
```

Deselect list line(s)

Command group	Flag affected	Reversible	Execute on client
List lines	YES	NO	NO

Syntax

Deselect list line(s) ([*All lines*]) {*line-number* (*calculation*)}

Options

All lines	If specified, the command affects all the lines in the list

Deprecated Command

This command has been deprecated and is no longer visible in the Code Assistant in the Code Editor (it will not appear when you type the first few characters), although it is still present in Omnis Studio and will continue to function if used in legacy code. You can show this command by disabling the appropriate Command Filter in the **Modify** menu in the Code Editor.

Description

This command deselects the specified list line. The specified line of the current list is deselected and is shown without highlight on a window list field when redrawn. You can specify the line number as a calculation. The *All lines* option deselects all lines of the current list. When a list is saved in the data file, the line selection state is stored.

Example

```
# Build a list and deselect line 5
Set current list lMyList
Define list {lCol1}
For lCount from 1 to 10 step 1
  Add line to list {(lCount)}
End For
Select list line(s) (All lines)
Deselect list line(s) {(lMyList.$linecount/2)}
# Alternatively, you can deselect a line by assigning its $selected property.
Do lMyList.5.$selected.$assign(kFalse) ## select line 5
```

Disable all menus and toolbars

Command group	Flag affected	Reversible	Execute on client
Menus	NO	YES	NO

Syntax

Disable all menus and toolbars

Description

The Disable all menus and toolbars command disables all top-level menus and toolbars in the main Omnis menu bar or application docking areas (i.e. not toolbars or menus installed in window classes). For toolbars, the command sets the active state of the docking areas and disables everything without changing its appearance. The menus and toolbars can be enabled using the Enable all menus and toolbars command.

Note $itoolbars represents the group of open top-level toolbar instances, and the members of the group are toolbars. Setting $enabled for an individual toolbar not only affects if controls can be used, but also affects the appearance of controls, hence they gray in this case.

You can disable all user installed menu and toolbar instances by setting the $enabled property, as follows:

```
Do $imenus.$sendall($ref.$enabled.$assign(kFalse))
Do $itoolbars.$sendall($ref.$enabled.$assign(kFalse))
```

Example

```
# Disable all menus and toolbars unless
# the correct password is entered
Disable all menus and toolbars
Prompt for input Password : Returns lPassword
If low(lPassword)='password'
  Enable all menus and toolbars
End If
```

Disable cancel test at loops

Command group	Flag affected	Reversible	Execute on clien
Constructs	NO	YES	NO

Syntax

Disable cancel test at loops

Description

Normally, Omnis tests if the user wishes to cancel execution of the method, at the end of each loop and during lengthy operations such as searching or sorting a large list. The user requests a cancel by either clicking on a working message Cancel button, or by pressing Ctrl-Break under Windows, Ctrl-C under Linux, or Cmnd-period under macOS. Use this command to disable these tests, meaning that the cancel key combination and clicks on a working message cancel button will be ignored.

This command is reversed with Enable cancel test at loops, or if placed in a reversible block.

Example

```
# delete all overdrawn accounts without interruption by the user requesting a cancel
Set main file {fAccounts}
Set search as calculation {fAccounts.Balance<0}
Find on fAccounts.Code (Use search)
Disable cancel test at loops
While flag true
  Working message (Repeat count) {Deleting Account [fAccounts.Code]}
  Delete
  Next on fAccounts.Code (Exact match)
End While
```

Disable enter & escape keys

Command group	Flag affected	Reversible	Execute on client	P
Enter data	NO	YES	NO	A

Syntax

Disable enter & escape keys

Description

This command disables the Enter key on all platforms; on Windows and Linux, it also disables the Escape key, whereas on macOS it also disables the Escape key and Cmnd-period. In other words, it disables the keyboard equivalents of the OK and Cancel pushbuttons. For example, you can use it during enter data mode to prevent the user from prematurely updating records by hitting the Enter key, when they attempt to start a new line. The option will remain set until either it is reversed with an Enable command, a new library is selected, or it is reversed as part of a reversible block.

Before using this command in a method that initiates an Enter data command, ensure that the user has some way of ending data entry, that is, by installing an OK and a Cancel pushbutton, or by using a $control() method that detects the end of data entry.

Example

```
# $construct of window class
Begin reversible block
  Disable enter & escape keys
End reversible block
Enter data
If flag true
  OK message {OK Button Pressed}
End If
```

Disable fields

Command group	Flag affected	Reversible	Execute on client
Fields	NO	NO	NO

Syntax

Disable fields {*list-of-field-names* (Name1,Name2,...)}

Description

This command disables the specified field or list of fields, making them inactive during Enter data and Prompted find. Thus the data entry cursor skips a disabled entry field when in data entry mode, find, and so on, and disabled pushbuttons cannot be clicked. If an entry field with scroll bar is disabled, you can tab to it but not change the data. You can reverse **Disable fields** or enable a display field using Enable fields.

Example

```
# disable 2 fields
Begin reversible block
  Disable fields {myField1,myField2}
End reversible block
Do method CheckCredit
Quit method
# now this method ends and the fields are re-enabled as they are in a reversible block
# to disable a single field on the current window
Do $cwind.$objs.myField1.$enabled.$assign(kFalse)
# to disable all fields on the current window like this
Do $cwind.$objs.$sendall($ref.$enabled.$assign(kFalse))
```

Disable menu line

Command group	Flag affected	Reversible	Execute on client
Menus	NO	YES	NO

Syntax

Disable menu line *line* or *instance-name/line*

Description

This command disables the specified line of a menu instance, that is, the menu line becomes grayed out and cannot be selected. You specify the menu-instance-name and the number of the menu line you want to disable. You can disable a complete menu instance by disabling line zero, that is the menu title.

You can reverse **Disable menu line** with the Enable menu line command or, you can use it in a reversible block. Nothing happens if the specified menu instance is not installed on the menu bar.

Example

```
# Install the menu mView and disable a menu line,
# the reversible block causes the menu line to be
# re-enabled when the method has finished
Install menu mView
Begin reversible block
  Disable menu line mView/Large
End reversible block
# Alternatively, you can set the $enabled property of a
# menu line using notation
Do $menus.mView.$obj.Large.$enabled(kFalse)
```

Disable relational finds

Command group	Flag affected	Reversible	Execute on c
Finding data	NO	YES	NO

Syntax

Disable relational finds

Description

This command reverses the action of Enable relational finds. The default situation is reinstated, that is, the main file and its connected parent files are joined using the Omnis connection.

Example

```
# Build a sorted combined list of parent and child data
# using an existing omnis connection
Disable relational finds ## this is the default action
Set main file {fChild}
Set current list lMyList
Define list {fChild,fParent}
Set sort field fParent.ID
Build list from file (Use sort)
```

Do

Command group	Flag affected	Reversible	Execute on c
Calculations	NO	NO	YES

Syntax

Do calculation **Returns** return-value

Description

This command executes the specified calculation, which is typically some notation that operates on a particular object or part of your library. It returns a value if you specify a *return-value,* which can be a variable of any type.

Note that where the return field is an item reference, the command sets the reference but does not assign to it: you must do this with Calculate or Do Itemref.$assign(value).

Example

```
# open a new window instance of the window class wMyWindow maximized
Do $clib.$windows.wMyWindow.$open('*',kWindowMaximize)

# redraw the current window instance
Do $cwind.$redraw()

# redraw EntryField1 on the top window
Do $topwind.$objs.EntryField1.$redraw()

# return a list in the local variable lClassList of all classes in the current library
Do $clib.$classes.$makelist($ref.$name) Returns lClassList

# close all open window instances
```

```
Do $iwindows.$sendall($ref.$close())

# set the $textcolor property of the current object to red
# the optional return field can be used to check whether the operation succeeded
Do $cobj.textcolor.$assign(kRed) Returns lFlag
```

Do async method

Command group	Flag affected	Reversible	Execute on client
Methods	NO	NO	NO

Syntax

Do async method remote-task-class/method-name (parameters) **Returns** return-value

Description

This command uses the Web Services server to execute a method asynchronously in the background, while the user continues to work with the application. Because it uses the Web Services server, you can only use this command if you meet some serial number requirements: you need a Web edition serial number, and for the development version, a Web Services serial number.

This command runs the specified remote task method. The method must have a name allowed for a Web Service method, and it must be marked as a static Web Service method. The method will only execute in the background if you have executed the Start server command to start the multi-threaded Web Client server. In a runtime, **Do async method** generates a runtime error if you use it before you have called Start server. In the development version, you can omit the call to Start server if you wish to debug the method; in this case, the method executes in the foreground, as if it were a normal method call.

The *return-value* is a long integer that uniquely identifies the call to the method. This is referred to as the *asynchronous call id*. You use the *asynchronous call id* to cancel the asynchronous method with the Cancel async method command, and to associate the completion message (see below) with the method call.

Passing Parameters

You can include a list of parameters with the **Do async method** command which are passed to the called method. If the called method has fewer parameters than values passed to it, the extra values are ignored.

Completion Message

When the method executing in the background finishes, Omnis sends a message to the task instance that was current when **Do async method** was called. The message is

$asynccomplete(iCallId,cErrText,vRetVal)

where iCallId is the asynchronous call id returned by **Do async method**, and vRetVal is the return value of the method executed in the background, unless an error occurred, in which case cErrText is not empty, and contains information about the error.

Notes

You can only call **Do async method** when running in the normal foreground thread.

Background threads pend while a message box is displayed.

The background threads only execute when the normal foreground thread is not executing.

The usual restrictions about remote task threads apply, for example you cannot debug a background thread, and you cannot use certain commands when running code in a background thread.

Execution of the remote task method occurs in the context of a remote task instance as usual. This means that the remote task $construct and $destruct methods are called before and after calling the specified method, and that the user count for the Web Client server must have an available connection.

If the library containing the remote task closes before the method finishes, Omnis stops its execution, and does not send the completion message. Note that if the method is in a critical block, Omnis will not stop its execution until it leaves the critical block. Also, execution will only stop after the current command being executed by the method completes.

Only use critical blocks for very short time periods in asynchronous methods, as the user interface will be unresponsive while code is running in a critical block.

Example

```
# Run the method $backgroundmethod asynchronously in the background - it prints a report, which the completion
# Returned long integer iCallId uniquely identifies the method call
Do async method REMOTETASK/$backgroundmethod ('rReport') Returns iCallId
# $backgroundmethod (implemented in remote task, and marked as a Web Service static method):
# Print the report identified by the parameter to memory, and return the resulting report
Calculate $devices.Memory.$visible as kTrue
Do $cdevice.$assign(kDevMemory)
Do $prefs.$reportdataname.$assign(iReport)
Set report name [pReportName]
# Note that Print report can be used in the multi-threaded Web Client server from Studio 4.1.5 onwards
Print report
Quit method iReport
# $asynccomplete(pCallId,pErrorText,pReport) in the task instance that was current when Do async method was ca
If len(pErrorText)=0
  Send to screen
  Print report from memory pReport
End If
```

Do code method

Command group	Flag affected	Reversible	Execute on client
Methods	NO	NO	NO

Syntax

Do code method code-class/method-name (parameters) **Returns** return-value

Description

This command runs the specified code class method, and accepts a value back from the called method. The specified *method-name* must be in the code class *code-class*. The command accepts a value back from the called method if you specify a *return-value*. The return field can be a variable of any type.

When a code class method is executed using this command, control is passed to the called method but the value of $cinst is unchanged, therefore the code in the code class method can refer to $cinst. When the code class method has executed, control passes back to the original executing method. The current task is not affected by execution moving to the code class.

Passing Parameters

You can include a list of parameters with the **Do code method** command which are passed to the called method. If the called method has fewer parameters than values passed to it, the extra values are ignored.

Note that where the return field is an item reference, the command sets the reference but does not assign to it: you must do this with Calculate or Do Itemref.$assign(value).

Example

```
# Call the method myMethod in the code class
# myCodeClass on a click event and pass the
# value of iMyVar as a parameter
On evClick
  Calculate iMyVar as 100
  Do code method myCodeClass/myMethod (iMyVar)
```

Do default

Command group	Flag affected	Reversible	Execute on
Calculations	YES	NO	NO

Syntax

Do default Returns *return-value*

Description

This command is used within the code for a custom property, and performs the default behavior for the built-in property with the same name as a custom property. **Do default** sets the flag if some built-in processing for the property exists.

Note that where the return field is an item reference, the command sets the reference but does not assign to it: you must do this with Calculate or Do Itemref.$assign(value).

Example

```
# Adding a method called $horzscroll.$assign to a window causes this method to be executed whenever
# Do $horzscroll.$assign is called. If the window is over 20 pixels wide when the method is called the default
# behavior for $horzscroll.$assign is performed, that is a scroll bar is added.
# declare parameter pScrollBarOn of type Boolean
If pScrollBarOn&$cinst.$width<20
  # window too narrow for a scroll bar
  Quit method
Else
  # assign a horz scroll bar
  Do default
End If
```

Do inherited

Command group	Flag affected	Reversible	Execute on
Calculations	YES	NO	YES

Syntax

Do inherited Returns *return-value*

Description

This command runs the superclass method with the same name as the currently executing method in the current subclass. For example, you can use **Do inherited** in the $construct() method of a subclass to execute the $construct() method of its superclass. Similarly you can run the $destruct() method in a superclass from a subclass.

The flag is set if a method with the name of the current method is found in one of the superclasses.

Example

```
# $construct method
Do inherited ## do superclass construct

# $destruct method
Do inherited ## do superclass destruct

# a method in a superclass can also be called using the $inherited method
Do $inherited.$mymethod
```

Do method

Command group	Flag affected	Reversible	Execute on client
Methods	NO	NO	YES

Syntax

Do method method-name (parameters) **Returns** return-value

Description

This command runs the specified method in the current class, and accepts a value back from the called method. If you use the **Do method** command in a field or line method, Omnis searches for the specified method in the field or line methods for the class, and then searches in the class methods. If the specified method is not found there is an error.

The command accepts a value back from the recipient or receiving method if you specify a return-value, which can be a variable of any type. Note that where the return field is an item reference, the command sets the reference but does not assign to it: you must do this with Calculate or Do *Itemref.$assign(value)*.

When another method is executed using this command, control is passed to the called method. When the called method has executed, control passes back to the original executing method. Note that you should use Do code method if you want to run a method in a code class, that is, a method outside the current class.

Passing Parameters

You can include a list of parameters with **Do method** which are passed to the called method. The parameters are taken in the order they appear in the parameter list and placed in the parameter variables in the called method. You can pass a reference to a field by using the special parameter variable type Field reference. This means that the called method can make changes to the field passed to it.

Recursion

Omnis allows a method to call itself, but will eventually run out of stack if the recursion does not terminate, or becomes too deep.

Example

```
# Call the method myMethod in the current instance which
# returns a value into iMyVar using Quit method lReturnValue
Do method myMethod Returns iMyVar
# Call myMethod and pass the field reference iMyFieldRef
# so that the value of iMyFieldRef can be changed by the
# method called
Calculate iMyFieldRef as 10
Do method myMethod (iMyFieldRef)
# You can use $cinst, $cfield, and $ctask to specify a method
# in the current instance, field, or task.
Do method $cinst.$mymethod
Do method $cfield.$myfieldmethod
Do method $ctask.$mytaskmethod
# You can also use the do command to call a method
Do $cinst.$mymethod
```

Do not flush data

Command group	Flag affected	Reversible	Execute c
Changing data	YES	YES	NO

Syntax

Do not flush data

Description

This command causes all data file operations to be carried out without writing the changed data to disk at each Update files or Delete. The command is designed to speed up data file operations when the user is prepared to take the extra risk of data loss.

The command operates best when there is a single user logged into the data file. It is unlikely to cause speed increase if the data is on a network volume (that is, shared by several users).

If you use Test for only one user at the beginning of the method, further users are prevented from opening the data file until the method terminates.

The command sets the flag if the state of the 'Do not flush data' mode is changed. When placed in a reversible block, the command restores the previous state of the 'Do not flush' flag upon the termination of the method.

Example

```
# fast import
Test for only one user
If flag true
  Do not flush data
  Drop indexes
End If

Prompt for import file
Prepare for import from file {Delimited(tabs)}
Import data lImportList
End import
Close import file

For each line in list from 1 to lImportList.$linecount step 1
  Prepare for insert ## transfer list to file
  Load from list
  Update files
End For
Flush data now ## writes the data immediately to disk
Build indexes ## rebuild indexes
Flush data ## Changes mode back to 'Flush data'
```

Do not wait for semaphores

Command group	Flag affected	Reversible	Execute c
Changing data	NO	YES	NO

Syntax

Do not wait for semaphores

Description

This command causes all commands which set semaphores to return with a flag clear if the semaphore is not available.

If **Do not wait for semaphores** is run first in a method, it will ensure that any subsequent commands that lock records, such as Prepare for..., Update commands, do not wait for records to be released. It causes the command to return a flag false and control to return immediately to the method, if a record is locked.

Semaphores

Semaphores are internal flags or indicators set in the data file to show other users that the record has been required elsewhere for editing. Semaphores are only set when running in multi-user mode, that is, the data file is located on a networked server, a Mac volume or on a DOS machine on which SHARE has been run.

The commands which set semaphores are Prepare for edit, Prepare for insert, Update files and Delete, and also, if prepare for update mode is on and the file acted upon is Read/Write, Single file find, Load connected records, Set read/write files, all types of Find, Next, and Previous. Update files commands lock the whole data file while indexes are re-sorted.

The Edit/Insert commands always wait for a semaphore, as do automatic find entry fields.

The example below illustrates how any command which causes a change in record locking requirements can fail (returning flag false). If, when in 'Prepare for' mode, a Single file find cannot lock the new record, it returns a flag false. This could mean either that the record could not be found, or that it was in use by another workstation. For this reason, it was made read-only before the Single file find and then changed to read/write. Note also that Update files can fail if the file cannot be locked while the indexes are re-sorted.

Example

```
Do not wait for semaphores
Prepare for edit
If flag true
  Set read-only files {fAccounts}
  Single file find on fAccounts.Code (Exact match)
  If flag false
    Cancel prepare for update
    Quit method kFalse
  End If
  Repeat
    Set read/write files {fAccounts}
  Until flag true
  Repeat
    Update files
  Until flag true
End If
```

Do redirect

Command group	Flag affected	Reversible	Execute on c
Calculations	YES	NO	NO

Syntax

Do redirect notation-for-object **Returns** return-value

Description

This command redirects execution from a custom property to any other public method. You specify the notation (or a calculation which evaluates to a reference to an object) for the recipient. The recipient of the custom property being processed is $crecipient. The flag is set if the recipient exists and handles the property with a built-in or custom property.

Example

```
Do $cwind.$setup ## the call to $setup in current window instance ..

# $setup method of the window instance
Do redirect $cwind.$objs.EntryField ## .. is diverted ..

# $setup method of EntryField ## .. to here
OK message {redirected to [$crecipient().$name]}
```

Does file exist

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

Does file exist (*file*|*folder-name*) **Returns** *err-code*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This command returns kTrue if the specified file or folder exists, otherwise it returns kFalse. The file or folder must specify the full path.

When constructing the path to a file or folder, you can use sys(9) to insert the correct path delimiter for the current platform: \ (back-slash) on Windows, or / (forward-slash) for Unix and 64-bit macOS (: colon on 32-bit macOS). In addition, you can use sys(115) to return the full pathname of the folder containing the Omnis executable, including the terminating path separator, which might be useful to reference files in the Omnis tree.

See also, the command Test if file exists.

Example

```
Calculate lPathname as con(sys(115),'html',sys(9),'serverusagetask.htm')
Calculate lNewPath as con(sys(115),'html',sys(9),'serverusagetask2.htm')
Copy file (lPathname,lNewPath) Returns lErrCode ## copies the file in lPathName to the filename contained in l
Does file exist (lNewPath) Returns lStatus ## see if the file exists
If lStatus
  Delete file (lNewPath) Returns lErrCode ## delete it
End If
```

Drop indexes

Command group	Flag affected	Reversible	Ex
Data management	YES	NO	NO

Syntax

Drop indexes {*file-name*}

Description

This command deletes all the indexes for the specified file apart from the record sequence number index. This enables intensive operations such as data import to proceed without the overhead of updating all the indexes. You can use Build indexes to rebuild the indexes which were dropped.

If the specified file name does not include a data file name as part of the notation, the default data file for that file is assumed. If the file is closed or memory-only, the command does not execute and returns with the flag false.

If you are running on a shareable volume, Omnis automatically tests that only one user is logged onto the data file (the command fails with flag false if this is not true) and further users are prevented from logging onto the data until the command completes.

If a working message with a count is open while the command is executing, the count will be incremented at regular intervals. The command may take a long time to execute, and it is not possible to cancel execution even if a working message with cancel box is open.

The command is not reversible: it sets the flag if it completes successfully and clears it otherwise, for example if there is more than one user logged onto the data file.

Example

```
# fast import
Do not flush data
Drop indexes {fCustomers} ## drop the indexes
Do method ImportData ## import the data
Build indexes {fCustomers} ## rebuild the indexes
```

Duplicate class

Command group	Flag affected	Reversible	Execute on client
Classes	YES	NO	NO

Syntax

Duplicate class {*class-name/new-name*}

Description

This command creates a new library class by duplicating an existing one. The name for the new class is specified in addition to the class you want to duplicate. Errors, such as attempting to use a name that is already in use, simply clear the flag and display an error message.

Typical uses of this command are to allow users to make changes to reports and searches.

Example

```
Duplicate class {sArea/sUser}
If flag true
  Modify class {sUser}
  Set search name sUser
  Print report (Use search)
End If
```

Else

Command group	Flag affected	Reversible	Execute on client
Constructs	NO	NO	YES

Syntax

Else

Description

This command is used after an If command to mark the beginning of some commands that are carried out if the condition in the preceding If command is false.

Example

```
# In the example below, the value of lGender is tested against the condition
# specified in the If statement. If the condition fails, control branches to the
# first Else If statement in the method. If the condition again fails, control
# branches to the Else command.
If lGender='M'
  OK message {Record is MALE}
Else If lGender='F'
  OK message {Record is FEMALE}
Else
  OK message (Sound bell ) {GENDER Unknown for this record}
End If
# The same result could also be obtained using a switch statement
Switch lGender
  Case 'M'
    OK message {Record is MALE}
  Case 'F'
    OK message {Record is FEMALE}
  Default
    OK message (Sound bell ) {GENDER Unknown for this record}
End Switch
```

Else If calculation

Command group	Flag affected	Reversible	Execute on client
Constructs	NO	NO	YES

Syntax

Else If calculation

Description

This command is used after an If command to mark the beginning of some commands that are carried out if the condition in the preceding If command is false, or the calculation in the Else If command is true.

Else If flag false

Command group	Flag affected	Reversible	Execute on client
Constructs	NO	NO	YES

Syntax

Else If flag false

Description

This command is used after an *If* statement and provides a marker before a series of commands that have to be carried out if the flag is false.

Example

```
# In the example below, the value of lGender is tested against the condition
# false if cancel if pressed.
Prompt for input Please enter your nam Returns lName (Cancel button)
If flag true
  OK message {Your name is [lName]}
Else If flag false ## cancel button pressed
  OK message {No name entered}
End If
```

Else If flag true

Command group	Flag affected	Reversible	Execute on client
Constructs	NO	NO	YES

Syntax

Else If flag true

Description

This command follows an If statement and provides a marker before a series of commands that have to be carried out if the flag is true and if the value does not meet the condition specified in the If statement.

Example

```
# use the Yes/No message to set or clear the flag
Yes/No message {Set flag with Yes or No}
If flag false
  OK message {flag is 0}
Else If flag true
  OK message {flag is 1}
End If
```

Enable all menus and toolbars

Command group	Flag affected	Reversible	Execute on client
Menus	NO	YES	NO

Syntax

Enable all menus and toolbars

Description

This command enables all top-level menus and toolbars in the main Omnis menu bar or application docking areas (i.e. not toolbars or menus installed in window classes). It reverses the action of Disable all menus and toolbars. This command will not enable a menu which has been disabled by disabling line zero. Such a menu can only be enabled by enabling line zero.

Example

```
# Enable all menus and toolbars if the correct
# password is enterd
Disable all menus and toolbars
Prompt for input Password : Returns lPassword
If low(lPassword)='password'
  Enable all menus and toolbars
End If
# Alternatively, you can enable all user installed menu
# and toolbar instances by setting the $enabled property
Do $imenus.$sendall($ref.$enabled.$assign(kTrue))
Do $itoolbars.$sendall($ref.$enabled.$assign(kTrue))
```

Enable cancel test at loops

Command group	Flag affected	Reversible	Execute on client
Constructs	NO	YES	NO

Syntax

Enable cancel test at loops

Description

This command causes Omnis to test if the user wishes to cancel execution of the method, at the end of each loop and during lengthy operations such as searching or sorting a large list. The user requests a cancel by either clicking on a working message Cancel button, or by pressing Ctrl-Break under Windows, Ctrl-C under Linux, or Cmnd-period under macOS. This command reverses the Disable cancel test at loops command. Unless Omnis has executed a Disable cancel test at loops, cancel testing is carried out automatically.

Example

```
# delete all overdrawn accounts without interruption by the user requesting a cancel
Set main file {fAccounts}
Set search as calculation {fAccounts.Balance<0}
Find on fAccounts.Code (Use search)
Disable cancel test at loops
While flag true
  Working message (Repeat count) {Deleting Account [fAccounts.Code]}
  Delete
  Next on fAccounts.Code (Exact match)
End While
Enable cancel test at loops ## enable break key for next loop
```

Enable enter & escape keys

Command group	Flag affected	Reversible	Execute on client
Enter data	NO	YES	NO

Syntax

Enable enter & escape keys

Description

This command enables the Enter key on all platforms; on Windows and Linux, it also enables the Escape key, whereas on macOS it also enables the Escape key and Cmnd-period. It reverses the action of the Disable enter & escape keys command.

In some libraries where the user may accidentally press Enter and terminate enter data mode, it is useful to disable the Enter key.

Example

```
# $construct of window class
Disable enter & escape keys
Enter data
If flag true
  OK message {OK Button Pressed}
End If
Enable enter & escape keys
```

Enable fields

Command group	Flag affected	Reversible	Execute on client
Fields	NO	NO	NO

Syntax

Enable fields {*list-of-field-names* (Name1,Name2,...)}

Description

This command enables the specified field or list of fields. You can use it to reverse the Disable fields command, or turn Display fields into Entry fields temporarily.

Example

```
# enable 2 fields
Begin reversible block
  Enable fields {myField1,myField2}
End reversible block
Prepare for insert
Enter data
Update files if flag set
Quit method
# now this method ends and the fields are re-disabled as they are in a reversible block
# to enable a single field on the current window
Do $cwind.$objs.myField1.$enabled.$assign(kTrue)
# to enable all fields on the current window like this
Do $cwind.$objs.$sendall($ref.$enabled.$assign(kTrue))
```

Enable menu line

Command group	Flag affected	Reversible	Execute on client
Menus	NO	YES	NO

Syntax

Enable menu line *line* or *instance-name*/*line*

Description

This command enables the specified line of a menu instance. It reverses the Disable menu line command. However, you cannot enable a line using this command if you have no access to it, or if there is no current record. You specify the menu-instance-name and the number of the menu line you want to enable. The command clears the flag if the menu instance is not installed or if the line cannot be enabled.

Example

```
# Install the menu mView and enable the menu line
# 'Large' if it is currently disabled
Install menu mView
Disable menu line mView/Large
Test for menu line enabled mView/Large
If flag false
  Enable menu line mView/Large
End If
```

Enable relational finds

Command group	Flag affected	Reversible	Execute on c
Finding data	NO	YES	NO

Syntax

Enable relational finds ([*Use connections*]) {*list-of-files* (F1,F2,..,Fn)}

Options

Use connections	If specified, all connections between the joined files are made when building the table

Description

This command causes all find tables to be built relationally, ignoring the main file. The file list is a list of files to be joined and, if Use connections is checked, all connections between the joined files are made when building the table. In effect, the connections provide the relational joins, that is, “sequence number = sequence number”.

When relational finds are enabled, the index field specified for find and build list commands is ignored. It is necessary to use a sort to determine the order of the table.

The Disable relational finds command causes a reversion to the default situation where the main file and its connected parent files are joined using the connections. The Enable relational finds and Disable relational finds commands are both reversible and do not affect the flag.

Example

```
Set current list lMyList
Define list {fChild,fParent,fGrandParent}
# Build a relational child/parent/grandparent list using omnis connections
Enable relational finds (Use con) {fChild,fParent,fGrandparent}
Build list from file
# Build a relational list of records ignoring omnis connections from fParent
# and fChild of parents with children less than 4 years old
Set search as calculation {fParent.ID=fChild.Parent_ID&fChild.Age<4}
Enable relational finds {fParent,fChild}
Build list from file (Use search)
```

Enclose exported text in quotes

Command group	Flag affected	Reversi
Importing and Exporting	NO	NO

Syntax

Enclose exported text in quotes ([*Enable*])

Options

Enable	If specified, all text exported in tab, comma and user delimited format is enclosed in quotes; executing the command without this option specified will cause text to be exported without quotes

Description

Example

```
Set report name rMyReport
Send to file
Prompt for print or export file
Enclose exported text in quotes (Enable)
Print report
# or disable the option with the notation
Do $clib.$prefs.$exportedquotes.$assign(kFalse)
```

End critical block

Command group	Flag affected	Reversible	Execute on client
Threads	NO	NO	NO

Syntax

End critical block

Description

End critical block is only applicable to the multithreaded server. It marks the end of a critical block.

See Begin critical block for more information on critical blocks.

Example

```
Begin critical block
  Calculate cClassVar as $cinst.$getvalue()
End critical block
```

End export

Command group	Flag affected	Reversi
Importing and Exporting	NO	NO

Syntax

End export

Description

This command ends the export of data from an Omnis list or row variable.

Example

```
# export to a file called myExport.txt in the root of your omnis tree
Calculate lExportPath as con(sys(115),'myExport.txt')
Set print or export file name {[lExportPath]}
Prepare for export to file {Delimited (commas)}
Export data lExportList
End export
Close print or export file
```

End For

Command group	Flag affected	Reversible	Execute on client
Constructs	NO	NO	YES

Syntax

End For

Description

This command ends a For loop. The two For loops For field value and For each line in list perform looping type operations. The End For command terminates both these commands.

Example

```
Do iMyList.$define(iMyCol1)
Do iMyList.$add('A')
Do iMyList.$add('B')
For iMyList.$line from 1 to iMyList.$linecount step 1
  Do iMyList.$loadcols()
  OK message {Line [iMyList.$line] = [iMyCol1]}
End For
Set current list iMyList
For each line in list from 1 to #LN step 1
  Load from list
  OK message {Line [iMyList.$line] = [iMyCol1]}
End For
```

End If

Command group	Flag affected	Reversible	Execute on client
Constructs	NO	NO	YES

Syntax

End If

Description

This command terminates an If statement once Omnis has executed the commands inside the If statement; it also marks the end of the commands to be executed as part of the If...Else Ifblock. Once the commands associated with the If...Else If block have been executed, control passes to the next command after End If. For every If command, you should have a corresponding End If command.

Example

```
For lCount from 1 to 100 step 1
  If lCount>=25&lCount<=50
    If lCount=25
      OK message {Quater of the way through now}
    Else If lCount=50
      OK message {Halfway through now}
    End If
  End If
End For
OK message {Done}
```

End import

Command group	Flag affected	Reversi
Importing and Exporting	NO	NO

Syntax

End import

Description

This command ends the import of data without closing the port, DDE channel, or file through which data is being imported.

Example

```
Prompt for import file
Prepare for import from file {Delimited (commas)}
Import data lImportList
End import
Close import file
```

End print

Command group	Flag affected	Reversible
Reports and Printing	YES	NO

Syntax

End print {instance-name}

Description

This command terminates the specified report and prints the totals section. If you omit the report instance name the End print command terminates the most recently started report instance. The flag is cleared if no report instances exist.

End print cancels the Prepare for print mode. You must include it after a Prepare for print command even if a totals section is not required.

You can print running totals of fields in the Record section by including the same fields in the Totals section of the report. Provided you choose the Totaled property for the field in the Record section, Omnis automatically maintains a running total.

Example

```
# Print report record by record
Set main file {fAccounts}
Set report name rMyReport
Send to screen
Prepare for print
Find first on fAccounts.Code
While flag true
  Print record
  Next
End While
End print
# Alternatively, you can end the print using notation
Do $ireports.rMyReport.$endprint()
```

End print job

Command group	Flag affected	Reversible
Reports and Printing	YES	NO

Syntax

End print job

Description

This command terminates a print job initiated with Begin print job and sends it to the printer.

End print job clears the flag and returns an error if a job has not been started. It sets the flag if it succeeds: in this case, the document is now available for the operating system to print.

Once a print job is started, any attempt to set the report destination fails, that is, you cannot select a new destination until you have issued an End print job.

Issuing End print job immediately after Begin print job may result in an empty document being printed.

Omnis automatically issues End print job at shutdown; it does not do this at any other time.

Example

```
# Create a print job and send 2 reports to the printer
Begin print job
Set report name rMyReport
Print report
Set report name rMyReport2
Print report
End print job
```

End reversible block

Command group	Flag affected	Reversible	Execute on client
Constructs	NO	NO	NO

Syntax

End reversible block

Description

This command defines the end of a reversible block of commands. All reversible commands enclosed within the commands Begin reversible block/End reversible block are reversed when the method containing this block finishes. However, a reversible block in the $construct() method of a window class reverses when the window is closed and not when the method is terminated as is normally the case.

Example

```
# A method can contain more than one block of reversible commands. In this case,
# commands contained within all the blocks are reversed when the method terminates.
# All the commands in the following example are reversed when the method containing
# the block is finished
Begin reversible block
  Disable menu line mMyMenu/5
  Set current list iMyList
  Build open window list (Clear list )
  Calculate iVar as 0
  Open window instance wMyWindow
End reversible block
# When this block is reversed:
# The window instance wMyWindow is closed
# iVar returns to its former value
# iMyList is restored to its former contents and definition
# The current list is set to the former value
# Menu line 5 is enabled
# The following method hides fields Entry1 and Entry2 and installs the menu mCustomers
Begin reversible block
  Hide fields {Entry1,Entry2}
  Install menu mCustomers
End reversible block
OK message (Icon) {MCUSTOMERS is now visible}
# When this method ends, first MCUSTOMERS is removed, then the fields are shown.
# In the following example, the current list is iMyList
Begin reversible block
  Set current list iMyList2
  Define list {fAccounts.Code,fAccounts.Surname,fAccounts.Balance}
```

```
  Set main file {fAccounts}
  Build list from select table
 Enter data
End reversible block
# When this method terminates and the command block is reversed, the Main file is reset,
# the former list definition is restored and the current list is restored to iMyList.
```

End statement

Command group	Flag affected	Reversible
SQL Object Commands	NO	NO

Syntax

End statement

Description

This command marks the end of a block of Sta: commands that build the SQL buffer for the current method stack. The Begin statement command defines the start of the block.

Example

```
# Open a multi-threaded omnis sql connection to
# the datafile mydatafile and create a statement to
# select rows from the table Customers
Calculate lHostname as con(sys(115),'mydatafile.df1')
Do iSessObj.$logon(lHostname,'','','MYSESSION')
Do iSessObj.$newstatement('MyStatement') Returns lStatObj
Begin statement
Sta: Select * From Customers
Sta: Where Cust_ID > 100
End statement
Do lStatObj.$execdirect()
Do lStatObj.$fetch(lMyList,kFetchAll)
```

End Switch

Command group	Flag affected	Reversible	Execute on client
Constructs	NO	NO	YES

Syntax

End Switch

Description

This command terminates a **Switch** statement and defines the point where method execution continues after each **Case** statement.

Example

```
# Select the correct graph window depending on the graph type selected in the pGraphType parameter.

# Declare Parameter GraphType (Short integer (0 to 255))
Switch pGraphType
  Case kGRpie
    Open window instance wGraphPieWindow
  Case kGRbars,kGRarea,kGRlines
    Open window instance wGraph2DWindow
  Case kGR3D
    Open window instance wGraph3DWindow
End Switch
```

End text block

Command group	Flag affected	Reversible	Execute on client
Text	NO	NO	YES

Syntax

End text block

Description

This command marks the end of a block of text which is placed in the text buffer for the current method stack. You build up the text block using the Begin text block and Text: commands. Following an End text block, you can return the contents of the text buffer using the Get text block command.

Example

```
Begin text block
Text: Thought for the day: (Carriage return)
Text: If a train station is where the train
Text: stops, what is a work station?
End text block
Get text block lTextString
OK message {[lTextString]}
```

End While

Command group	Flag affected	Reversible	Execute on client
Constructs	NO	NO	YES

Syntax

End While

Description

This command marks the end of a **While** loop. When the condition specified at the start of the loop is not fulfilled (testing the flag or calculation) the command after the **End While** command is executed. Each loop that begins with a **While** command must terminate with an **End While** command, otherwise an error occurs.

Example

```
Calculate lCount as 1
While lCount<=3 ## While loop
  Calculate lCount as lCount+1
End While
OK message {Count=[lCount]} ## prints 'Count=4'
Calculate lCount as 1
Repeat ## Repeat loop
  Calculate lCount as lCount+1
Until lCount>=3
OK message {Count=[lCount]} ## prints 'Count=3'
```

Enter data

Command group	Flag affected	Reversible	Execute on client
Enter data	YES	NO	NO

Syntax

Enter data until *termination-condition* (leave blank to terminate on OK or Cancel)

Description

This command puts Omnis into enter data mode which allows data to be entered via the current window. An error is generated if there is no open window. It initiates an internal control loop which does the following:

- Places the cursor in the first entry field,
- Lets the user enter data from the keyboard,
- Detects the use of Tab, Shift-Tab and other cursor movements such as click and moves the cursor to the appropriate field,
- Waits for an OK, setting flag true before allowing control to pass to the command following ***Enter data*** in the method,
- Detects a Cancel which aborts data entry with a false flag.

By default, the **Enter data** command waits for an evOK or evCancel event. When these events are triggered enter data mode is terminated (assuming the window is not in modeless enter data mode). However you can include a termination condition with **Enter data** which causes enter data mode to continue until the expression becomes true.

Example

```
# $construct of window class
Enter data ## waits for a evOK or evCancel event
If flag true
  OK message {User has pressed Return}
Else
  OK message {User has canceled}
End If
# or
# $construct of window class
Calculate iValue as 0
```

Export data

Command group	Flag affected	Reversi
Importing and Exporting	NO	NO

Syntax

Export data list-or-row-name

Description

This command exports data from an Omnis list or row variable.

Example

```
# export to a file called myExport.txt in the root of your omnis tree
Calculate lExportPath as con(sys(115),'myExport.txt')
Set print or export file name {[lExportPath]}
Prepare for export to file {Delimited (commas)}
Export data lExportList
End export
Close print or export file
```

Export fields

Command group	Flag affected	Reversi
Importing and Exporting	YES	NO

Syntax

Export fields index-name ([Indirect][,Use search][,Disable messages]) {list-of-field-names (Name1,Name2,...)}

Options

Indirect	If specified, the command uses the contents of the first field as the list of fields
Use search	If specified, the command uses the current search to select data
Disable messages	If specified, the command does not open messages requiring a user response and instead it writes a limit amount of information to the trace log

Description

Export fields exports the data for the list of fields to the current export file. It provides runtime access to the functionality of the export data dialog in the IDE. The command sets the main file for the export to the file corresponding to the first field in the list. The index-name is the optional name of the indexed field which determines the order of the exported data.

Example

```
# export to a file called myExport.txt in the root of your omnis tree
Calculate lExportPath as con(sys(115),'myExport.txt')
Set print or export file name {[lExportPath]}
Prepare for export to file {Delimited (commas)}
Export fields fCustomers.CustomerID {fCustomers.Surname,fCustomers.FirstName}
End export
Close print or export file
```

FileOps error codes

Error Code	Error Text
1	Too few parameters passed on the command line
12	Out of memory error
998	Undefined error
999	No operation on this platform
-30	Unable to delete directory or file
-36	Disk IO error (or error during operation)
-39	End of file reached during Read file as character or Read file as binary
-43	File not found
-48	File or directory already exists
-51	Bad file reference number
-59	Problem during rename

Find

Command group	Flag affected	Reversible	Execute on
Finding data	YES	YES	NO

Syntax

Find on field-name ([Exact match][,Use search]) {calculation}

Options

Exact match	If specified, the index value of the field in suitable records must equal the current value
Use search	If specified, the command uses the current search to select data

Description

This command builds a find table and locates the first record in the table, that is, it loads the main and connected files into the current record buffer. The flag is false and the buffer is cleared if no record is found.

You use the **Find** command to locate records within a file. If you don't use a search, the file is searched in the order specified by the indexed field until the value given in the calculation line is matched. In this case, the current find table is the same as the chosen Index.

When the closest match is found, the main and connected files are read into the current record buffer and the flag is set true. If the indexed field is from a connected file, the search is repeated automatically until the record having a connected entry in the main file is found.

A blank calculation indicates that the **Find** is to be performed using the current value of the selected index field. Thus, if you precede the command with a Clear main file, it is the same as a Find first.

Omnis can perform a **Find** with an Exact match requirement. In this case, the value in the "field found" record must correspond in every detail (for example, upper or lower case characters) to the current value of the indexed field in the current record buffer. A flag true indicates a successful Find, otherwise a flag false results, and the main and its connected files are cleared.

You use the exact match option to locate child records connected to a current parent record.

Clearing the find table

The find table is cleared if:

- A Clear find table command is executed with the same main file setting.
- A new Find is carried out on the same file.
- A Next/Previous command with a new (non-blank) index or a Use Search or Exact match option where the original Find had none, is used.

Example

```
# Find all invoices belonging to account lMyAccCode
Prompt for input Account Code ? Returns lMyAccCode (Cancel button)
If flag true
  Set main file {fInvoices}
  Set search as calculation {fInvoices.AccCode=lMyAccCode}
  Find on fInvoices.InvNum (Exact match,Use search)
  While flag true
    OK message {Found Invoice [fInvoices.InvNum] for account [fInvoices.AccCode]}
    Next
  End While
End If
```

Find first

Command group	Flag affected	Reversible	Execute on
Finding data	YES	YES	NO

Syntax

Find first on field-name ([Use search][,Use sort])

Options

Use search	If specified, the command uses the current search to select data
Use sort	If specified, the command uses the current sort field(s) to order the data

Description

This command automatically locates the first record in a file using the index for the specified field. If no field is given, the record sequence number is used. The main and connected files are read into the CRB if a valid first record is found. The flag is set false if no record is found.

You use the Use search option in conjunction with the specified indexed field to select the first record which fulfils the search specification. If the search is a calculation, the optimizer will choose the best index if the index field is left blank.

You use the Use Sort option in conjunction with the current sort fields (see Set sort field) to create a table of entries from the data file which are sorted into an order set by up to nine sort fields.

The find table is cleared if:

- A Clear find table command is executed with the same main file setting.
- A new Find is carried out on the same file.
- A Next/Previous command with a new (non-blank) index or a *Use Search* or *Exact match* option where the original *Find* had none, is used.

If you use the *Find first* command within a reversible block, it is reversed when the method finishes, that is, the main and connected records are restored. However, if the data within the original record has been deleted or changed, it will not be possible to completely restore the buffer.

Example

```
# Find the first account with a negative balance, but restore
# the original record when this method finishes
Begin reversible block
  Set main file {fAccounts}
  Set search as calculation {fAccounts.Balance<0}
  Find first on fAccounts.Code (Use search)
End reversible block
```

Find last

Command group	Flag affected	Reversible	Execute on
Finding data	YES	YES	NO

Syntax

Find last on field-name ([Use search][,Use sort])

Options

Use search	If specified, the command uses the current search to select data
Use sort	If specified, the command uses the current sort field(s) to order the data

Description

This command automatically locates and displays the last record in a file using a specified indexed field. You can use the Find last command to locate the last record added to a file by using the record sequencing number as the index. The flag is set false if no record is found.

You use the Use search option in conjunction with the specified indexed field to select the last record which fulfils the search specification. If the search is a calculation, the optimizer will choose the best index if the index field is left blank.

Whenever you use a Find command, a find table is created which determines the order in which records are displayed using subsequent Next and Previous commands. Once a find table has been created, subsequent Next or Previous commands will use the table provided the commands have an empty or the same Index, and the same (or empty) Search and Exact match conditions. A Clear find table, a new Find on the same file or Next/Previous commands with a new (non-blank) index or a Search or Exact match where the original Find had none, will clear the find table.

The Use Sort option works in conjunction with the current sort fields (see Set sort field) to create a table of entries from the data file which are sorted into an order set by up to 9 sort fields. Refer to the Find command for details of the find table and its use.

Example

```
# Find the last account record in the file, but restore
# the original record when this method finishes
Begin reversible block
  Set main file {fAccounts}
  Find last on fAccounts.Code (Use search)
End reversible block
```

Floating default data file

Command group	Flag affected	Reversible	Execute on client
Data files	NO	YES	NO

Syntax

Floating default data file {*list-of-files* (F1,F2,..,Fn)}

Description

This command sets the default data file as the current data file and changes whenever the current data file changes. You use Floating default data file in libraries which open more than one data file at once. The default behavior in Omnis is that, as each new data file is opened, it becomes the "current" data file. The concept of a current data file is important when your commands refer to file classes without specifying a data file.

The Floating default data file command sets the default data file, for the specified list of files, to be equal to the current data file and allows it to change (float) whenever the current data file changes.

The command does not change the flag but is reversible, that is, the previous default data files are restored when the method containing the command in a reversible block terminates.

Example

```
# To specify the data file, you can use Set Default Data File to associate a file class with the
# current data file. In this example we associate fCustomers with Data.df1
Set current data file {Data1}
Set default data file {fCustomers}
# References to fCustomers are now equivalent to references to Data1.fCustomers.
# The association between fCustomers and Data1 remains in effect even if the current data file
# is set to a different data file. To return to the default state where the default data file "floats"
# to whatever the current data file is, you can use:
Floating default data file {fCustomers}
```

Flush data

Command group	Flag affected	Reversible	Execute c
Changing data	YES	YES	NO

Syntax

Flush data

Description

This command reverses Do not flush data and reverts to the default mode where the changed data is immediately written to disk after each Update files or Delete command.

The command sets the flag if the state of the 'Do not flush data' mode is changed and is reversible, restoring the previous state of the 'Do not flush' flag when reversed. If the previous mode was 'Do not flush data', Flush data will cause any modified data which has not been written to disk, to be written on the next Update files or Delete.

Example

```
# fast import
Test for only one user
If flag true
  Do not flush data
```

```
  Drop indexes
End If
Prompt for import file
Prepare for import from file {Delimited(tabs)}
Import data lImportList
End import
Close import file
For each line in list from 1 to lImportList.$linecount step 1
  Prepare for insert ## transfer list to file
  Load from list
  Update files
End For
Flush data now ## writes the data immediately to disk
Build indexes ## rebuild indexes
Flush data ## Changes mode back to 'Flush data'
```

Flush data now

Command group	Flag affected	Reversible	Execute on client	P
Changing data	NO	NO	NO	A

Syntax

Flush data now

Description

This command causes any modified data which has not been written to disk to be immediately written to disk. This command will only do something if a Do not flush data command has been executed.

This command leaves the flag unaffected and is not reversible.

Example

```
# fast import
Test for only one user
If flag true
  Do not flush data
  Drop indexes
End If
Prompt for import file
Prepare for import from file {Delimited(tabs)}
Import data lImportList
End import
Close import file
For each line in list from 1 to lImportList.$linecount step 1
  Prepare for insert ## transfer list to file
  Load from list
  Update files
End For
Flush data now ## writes the data immediately to disk
Build indexes ## rebuild indexes
Flush data ## Changes mode back to 'Flush data'
```

For each line in list

Command group	Flag affected	Reversible	Execute on client
Constructs	NO	NO	NO

Syntax

For each line in list ([*Selected lines only*][,*Descending*]) **from** *start* **to** *stop* **step** *step*

Options

Selected lines only	If specified,the for loop only operates on the selected lines in the list
Descending	If specified,the for loop steps through the list from the largest line number to the smallest line numb

Description

This command marks the beginning of a loop that processes the lines of the current list. You must specify the current list before executing the For loop. The For loop is a convenient way to write While/ End While loops to step through each line of a list. With the Selected lines only option, the loop will skip over any lines encountered that are not selected.

The Start value specifies the line in the list at which method execution of the For loop starts. The loop continues until the processed line exceeds or is equal to the Stop value. If the Stepvalue is not specified, the default value of 1 is used. The values involved must all be integers. The Descending option tells Omnis to step through the list from a high line number to a low line number. The Start and Stop values are swapped if the Stop value is less than the Start value.

You can use Jump to start of loop within the loop to continue the next iteration of the loop. Similarly, Break to end of loop will exit the loop prematurely.

For each line in list operates on the current list. The matching End For will also operate on the current list. Unpredictable behavior will result if the current list is changed and not restored within the For/ End For construct.

Example

```
Prepare for print
Set current list iMyList
For each line in list from 1 to iMyList.$linecount step 1
  Load from list
  Print record
End For
End print
# this is equivalent to the method below
Prepare for print
Set current list iMyList
Calculate iMyList.$line as 1
While iMyList.$line<=iMyList.$linecount
  Load from list
  Print record
  Calculate iMyList.$line as iMyList.$line+1
End While
End print
```

For field value

Command group	Flag affected	Reversible	Execute on client
Constructs	NO	NO	YES

Syntax

For field-name **from** start **to** stop **step** step

Description

This command marks the beginning of a For loop which defines a series of commands to be repeated a number of times. You use field-name as a counter that is automatically incremented by the step value each time the End For statement is reached.

The values involved must all be numbers, preferably integers. If start value is greater than end value, and step value is positive, the command will perform no loops. Similarly, no loops are performed if start value is less than end value, and step value is negative.

The end value is evaluated once at the start of the loop, and saved, for performance reasons, so changing the end value during the loop will have no effect. You can use Jump to start of loop within the loop to continue the next iteration of the loop. Similarly, you can terminate the loop early using Break to end of loop if desired.

Example

```
Calculate lString as ''
For lCount from 0 to 9 step 1
  Calculate lString as con(lString,lCount)
End For
OK message {String=[lString]} ## shows 'String=0123456789'
Calculate lString as ''
For lCount from 9 to 0 step -1
  Calculate lString as con(lString,lCount)
End For
OK message {String=[lString]} ## shows 'String=9876543210'
```

FTPChmod

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

FTPChmod (socket,filename,mode) **Returns** status

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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 Linux-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. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# allow owner to read/write & execute, group to read & execute and world to read-only this file
Calculate lFileMode as 754
FTPChmod (iFTPSocket,lFileName,lFileMode) Returns lErrCode
```

FTPConnect

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

FTPConnect (*serveraddr*,*username*,*password*[,*port*,*errorprotocoltext*,*secure* {Default zero insecure;1 secure;2 use AUTH TLS},*verify* {Default kTrue}, *charset* {Default kUniTypeAuto}]) **Returns** *socket*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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 or pass an empty value, it defaults to the standard FTP port (21 for non-secure or AUTH TLS secure connections, or 990 for other secure connections). If you use a service name, the lookup for the service will occur locally.

ErrorProtocolText is an optional Omnis Character field parameter, into which **FTPGetConnect** places the protocol exchange that occurred on the control connection to the FTP server, if an error occurred. Note that you can use the command FTPGetLastStatus to obtain the protocol exchange in the case when a connection is successfully established.

Secure is an optional Boolean* parameter which indicates if a secure connection is required to the server. Pass kTrue for a secure connection, in which case the built-in security technology will be used, so on Windows 'Secure Channel' (Schannel) is used, on macOS 'Secure Transport' is used, and on Linux OpenSSL is used.

***FTPConnect** also supports an alternative *secure* option, if you pass *secure* with the value 2, the connection is initially not secure, but after the initial exchange with the server, **FTPConnect** issues an AUTH TLS FTP command to make the connection secure if the server supports it (see RFC 4217 for details), followed by further commands necessary to set up the secure connection. Authentication occurs after a successful AUTH TLS command.

Note that if you use either of the secure options, all data connections are also secure, and all data transfer uses passive FTP.

AUTH TLS is the standard recommended mechanism for FTPS, and is referred to as *explicit* FTPS. The other secure form of FTP supported by this command is referred to as *implicit* FTPS, and is no longer recommended; however, we provide support for implicit FTPS to cater for servers which do not support explicit FTPS.

FTPS resumes the TLS session for data connections. In addition, it automatically sends PBSZ and PROT commands to the server after establishing a secure control connection.

Verify is an optional Boolean parameter which is only significant when Secure is not kFalse. When Verify is kTrue, the command instructs the installed SSL library to verify the server's identity using its certificate; if the verification fails, then the connection will not be established. You can pass Verify as kFalse, to turn off this verification; in this case, the connection will still be encrypted, but there is a chance the server is an impostor. In order to perform the verification, the installed SSL library uses the Certificate Authority Certificates in the cacerts sub-folder of the secure folder in the Omnis folder. If you use your own Certificate Authority to self-sign certificates, you can place its certificate in the cacerts folder, and the installed SSL library will use it after you restart Omnis.

Charset specifies the character set used for exchanging pathnames with the FTP server, and for exchanging file character data with the FTP server. Charset can either be kUniTypeAuto, kUniTypeUTF8, kUniTypeNativeCharacters, kUniTypeAnsi..., kUniTypeISO8859_..., or kUniTypeOEM.

If you specify kUniTypeAuto, after **FTPConnect** establishes a connection, it sends a FEAT command to the server to determine if the server supports UTF8. If the server supports UTF8, then the connection uses UTF8 as the charset, otherwise it uses kUniTypeNativeCharacters.

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. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
FTPConnect (iServerAddress,iUserName,iPassword) Returns iFTPSocket
If iFTPSocket<0
  FTPGetLastStatus (iServerReplyText) Returns lErrCode
  OK message FTP Error {[con("An error occurred logging on to ",iServerAddress," - Details follow",kCr,iServer
End If
```

FTPCwd

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

FTPCwd (*socket*,*newdir*) **Returns** *status*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded,allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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 Linux-style path and file specifications with path and file separated by slashes, such as

/drive/user/subdirectory/filename.extension

Most FTP servers accept the Linux 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 Linux style.

Status is an Omnis Long Integer field which receives the result of executing the command. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
Calculate lNewDirectory as '../testFolder'

FTPCwd (iFTPSocket,lNewDirectory) Returns lErrCode
If lErrCode
  OK message FTP Error {[con("Error setting FTP directory",kCr,"Error code : ",lErrCode)]}
End If
```

FTPDelete

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

FTPDelete (*socket*,*filename*[,*directory* {Default kFalse}]) **Returns** *status*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded,allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

FTPDelete deletes a file or directory 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 or directory to delete.

Directory is an optional Boolean (that defaults to kFalse) which you pass as kTrue if Filename is the pathname of a directory rather than a file. Note that a directory may need to be empty before you can delete it.

Status is an Omnis Long Integer field which receives the result of executing the command. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
Calculate lFileName as 'myFileToDelete'
FTPDelete (iFTPSocket,lFileName) Returns lErrCode
If lErrCode
  OK message FTP Error {[con("Error deleting ",lFileName,kCr,"Status code: ",lErrCode)]}
End If
```

FTPDisconnect

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

FTPDisconnect (*socket*) **Returns** *status*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded,allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
FTPDisconnect (iFTPSocket) Returns lErrCode
If lErrCode
  OK message FTP Error {[con("Error disconnecting from FTP server ",kCr,"Error code : ",lErrCode)]}
End If
```

FTPGet

Command group	Flag affected	Reversible	Execute on client	P
External commands	YES	NO	NO	A

Syntax

FTPGet (socket,remotefile,localfile[,filetype,creator]) **Returns** status

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# set file transfer mode to asci
FTPType (iFTPSocket,0) Returns lErrCode
If not(lErrCode)
  # assumes you are already in the correct folder on the ftp server so only the file name is needed
  Calculate lRemoteFile as 'myFileToDownload.txt'
  # identify where to download the file to
  Calculate lLocalFileName as con(sys(115),'downloadFolder',sys(9),lRemoteFile)
  # download the file
  FTPGet (iFTPSocket,lRemoteFile,lLocalFileName) Returns lErrCode
  If lErrCode
    OK message FTP Error {[con("Error transferring file ",upp(lRemoteFile)," to ",upp(lLocalFileName),kCr,"Err
  End If
End If
```

FTPGetBinary

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

FTPGetBinary (socket,remotefile,binfield) **Returns** status

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# set file transfer mode to binary
FTPType (iFTPSocket,1) Returns lErrCode
If not(lErrCode)
  # assumes you are already in the correct folder on the ftp server so only the file name is needed
  Calculate lRemoteFile as 'omnis.exe'
  # download the file
  FTPGetBinary (iFTPSocket,lRemoteFile,lBinField) Returns lErrCode
  If lErrCode
    OK message FTP Error {[con("Error transferring file ",upp(lRemoteFile),"Error code : ",lErrCode)]}
  Else
    # select where to save the file to on the local machine
    Do FileOps.$selectdirectory(lNewPath,'Enter path to save file to',sys(115)) Returns lStatus
    If lStatus
      # create the file
      Do lFileOps.$createfile(con(lNewPath,sys(9),lRemoteFile))
      # write the binary contents downloaded from the FTP server to the new local file
      Do lFileOps.$writefile(lBinField)
    End If
  End If
End If
```

FTPGetLastStatus

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

FTPGetLastStatus (*socket*[,*protocoltext*]) **Returns** *status*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded,allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

FTPGetLastStatus returns status information corresponding to the last FTP command executed on a connected FTP socket.

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

ProtocolText is an optional Omnis Character field parameter, into which **FTPGetLastStatus** places the FTP protocol exchange that occurred on the control connection to the FTP server, for the last FTP command executed. For example, if you execute FTPPwd, and then call **FTPGetLastStatus**, ProtocolText 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. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# Example to show how to get the error message from the FTP server when the download fails
# set file transfer mode to asci
FTPType (iFTPSocket,0) Returns lErrCode
If not(lErrCode)
  # assumes you are already in the correct folder on the ftp server so only the file name is needed  Calculate
  # identify where to download the file to
  Calculate lLocalFileName as con(sys(115),'downloadFolder',sys(9),lRemoteFile)
  # download the file
  FTPGet (iFTPSocket,lRemoteFile,lLocalFileName) Returns lErrCode
  If lErrCode
    FTPGetLastStatus (iFTPSocket,iServerReplyText) Returns lErrCode
    OK message FTP Error {[con("Error transferring file ",
 upp(lRemoteFile),kCr,"Error text from the server: ",
 kCr,iServerReplyText)]}
  End If
End If
```

FTPList

Command group	Flag affected	Reversible	Execute on client	P
External commands	YES	NO	NO	A

Syntax

FTPList (socket,list[,pathname,mode]) **Returns** status

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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 Linux 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. Possible error codes are listed in the Web Command Error Codes Appendix

Example

```
Do iMyList.$define(iListColumn)
# return a long format listing of the current directory into the list variable iMyList
FTPList (iFTPSocket,iMyList,,1) Returns lErrCode
If lErrCode
  FTPGetLastStatus (iServerReplyText) Returns lErrCode
  OK message FTP Error {[con("Error obtaining list of files from the FTP server",kCr,"Details follow: ",kCr,iS
End If
```

FTPMkdir

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

FTPMkdir (*socket*,*dirname*) **Returns** *status*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# create a new directory called myNewDirectory in the directory Test
Calculate lDirName as '/Test/myNewDirectory'
FTPMkdir (iFTPSocket,lDirName) Returns lErrCode
If lErrCode
  FTPGetLastStatus (iServerReplyText) Returns lErrCode
  OK message FTP Error {[con("Error creating directory",lDirName,kCr,"Details follow: ",kCr,iServerReplyText)]
End If
```

FTPPut

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

FTPPut (socket,localfile,remotefile) **Returns** status

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# upload an ascii file to a FTP server
# set file transfer mode to asci
FTPType (iFTPSocket,0) Returns lErrCode
If not(lErrCode)
  Calculate lLocalFileName as con(sys(115),'uploadFolder',sys(9),'myTextFileToUpload.txt')
  Calculate lRemoteFile as 'myUploadedFile.txt'
  # upload the file to the current working directory on the FTP server, the file name will be myUploadedFile.t
  FTPPut (iFTPSocket,lLocalFileName,lRemoteFile) Returns lErrCode
  If lErrCode
    FTPGetLastStatus (iServerReplyText) Returns lErrCode
    OK message FTP Error {[con("Error uploading file",upp(lLocalFileName)," to ",upp(lRemoteFile),kCr,"Details
  End If
End If
```

FTPPutBinary

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

FTPPutBinary (socket,binfield,remotefile) **Returns** status

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# upload a binary file to a FTP server
# set file transfer mode to binary
FTPType (iFTPSocket,1) Returns lErrCode
If not(lErrCode)
  # select the binary file to upload
  Do FileOps.$getfilename(lDirName,'Select the binary file to upload','*.*',sys(115))
  Do lFileOps.$openfile(lDirName)
  # read contents into an Omnis binary variable
  Do lFileOps.$readfile(lBinField)
  Calculate lRemoteFile as '/Test/upload/myUploadedFile'
  FTPPutBinary (iFTPSocket,lBinField,lRemoteFile) Returns lErrCode
  If lErrCode
    FTPGetLastStatus (iServerReplyText) Returns lErrCode
    OK message FTP Error {[con("Error uploading binary",upp(lDirName)," to ",upp(lRemoteFile),kCr,"Details fol
  End If
End If
```

FTPPwd

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

FTPPwd (socket) **Returns** server-directory

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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. Possible error codes are listed in the Web Command Error Codes Appendix.

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

Example

```
# return the current working directory on the FTP server
FTPPwd (iFTPSocket) Returns lDirectory
If lDirectory<0 ;; an error has occurred
  FTPGetLastStatus (iServerReplyText) Returns lErrCode
  OK message FTP Error {[con("Error obtaining current FTP directory",kCr,"Details follow: ",kCr,iServerReplyTe
End If
```

FTPReceiveCommandReplyLine

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

FTPReceiveCommandReplyLine (*socket*) **Returns** *reply*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
FTPSendCommand (iFTPSocket,'pwd')
# return the current directory
FTPReceiveCommandReplyLine (iFTPSocket) Returns lDirName
```

FTPRename

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

FTPRename (*socket*,*oldname*,*newname*) **Returns** status

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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. Possible error codes are listed in the Web Command Error Codes Appendix.

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

Example

```
# rename a file or folder in the current working directory
FTPRename (iFTPSocket,lFileName,lNewFileName) Returns lErrCode
If lErrCode
  FTPGetLastStatus (iServerReplyText) Returns lErrCode
  OK message FTP Error {[con("Error renaming ",lFileName," to ",lNewFileName,kCr,"Details follow:",kCr,iServer
End If
```

FTPSendCommand

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

FTPSendCommand (*socket*,*command*) **Returns** *status*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
FTPSendCommand (iFTPSocket,'pwd')
# return the current directory
FTPReceiveCommandReplyLine (iFTPSocket) Returns lDirName
```

FTPSetConfig

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

FTPSetConfig (*proc*[,*activeonly* {Default zero for no;1 for yes}]) **Returns** *status*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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 over non-secure connections to be active, rather than the default, which is use passive FTP if the server supports it (if the connection is secure then only passive FTP can be used). 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.

Status receives the result of executing this command. Possible error codes are listed in the Web Command Error Codes Appendix.

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 *FTP-Put* or *FTPPutBinary*.

The FTP data transfer commands always first attempt to use passive mode to transfer data. In passive mode, the client initiates the 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.

Example

```
# setup the config method
FTPSetConfig ('cCode/FTPProgress')
# Then in code class cCode/FTPProgress
# 3 parameter variables (all defined as long integer)
OK message {Socket [pSocket] - TransferredSoFar [pTransferredSoFar] - TotalToTransfer [pTotalToTransfer]}
```

FTPSite

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

FTPSite (*socket*,*parameters*) Returns status

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# execute a FTP site specific command
FTPSite (iFTPSocket,'SITE CHMOD 744 /test/myFileToChange') Returns lErrText
```

FTPType

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

FTPType (*socket*,*filetype*) **Returns** *status*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# set file transfer mode to ascii
FTPType (iFTPSocket,0) Returns lErrCode
If not(lErrCode)
 # assumes you are already in the correct folder on the ftp server so only the file name is needed
 Calculate lRemoteFile as 'myFileToDownload.txt'
 # identify where to download the file to
 # here we decide to put the download file into a folder called downloadFolder within the current Omnis tree
 Calculate lLocalFileName as con(sys(115),'downloadFolder',sys(9),lRemoteFile)
 # download the file
 FTPGet (iFTPSocket,lRemoteFile,lLocalFileName) Returns lErrCode
 If lErrCode
 OK message FTP Error {[con("Error downloading file ",
 upp(lRemoteFile)," to ",upp(lLocalFileName),
 kCr,"Error code : ",lErrCode)]}
 End If
End If
```

Get file info

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

Get file info (*path, type, creator, log-size, phy-size, creat-date, creat-time, mod-date, mod-time*) **Returns** err-code

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This command returns information about the file specified in path.

A file may occupy more physical disk space than is necessary, because disk space is usually allocated in blocks of some fixed size. This is why the logical and physical sizes can be different.

It returns an error code (See Error Codes), or zero if no error occurs.

When constructing the path to a file or folder, you can use sys(9) to insert the correct path delimiter for the current platform: \ (back-slash) on Windows, or / (forward-slash) for Unix and 64-bit macOS (: colon on 32-bit macOS). In addition, you can use sys(115) to return the full pathname of the folder containing the Omnis executable, including the terminating path separator, which might be useful to reference files in the Omnis tree.

Example

```
# return the file info for the omnis executable
Calculate lFileName as con(sys(115),'omnis.exe')
Get file info (lFileName,lFileType,lFileCreator,lFileLogicalSize,lFilePhysicalSize,lFileCreationDate,lFileCrea
```

Get file name

Command group	Flag affected	Reversible	Execute on client	P
External commands	YES	NO	NO	V

Syntax

Get file name (*path* [,*dialog-title*] [,*file-type*...]) **Returns** *err-code*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This command opens the standard Open file dialog for the current Operating System, in order to obtain the path of a file selected by the user. You would typically use this command to prompt the user for the path of an existing file. If you want to prompt the user to enter the path of a new file, use the Put file name command instead.

You can specify a dialog-title for the Open dialog.

The optional file-type parameter limits the choice of file types available.

Get file name returns the full pathname of the file the user selects in path, or path remains empty if no file is selected (that is, the Cancel button was clicked). The selected file is not opened.

It returns an error code (See Error Codes), or zero if no error occurs.

File types

You can specify one or more extensions (using wildcard patterns like those used in many DOS and shell commands) separated by semicolons. For example, "*.TXT" would specify text files only.

Example

```
# open the Get File dialog and show only omnis libraries
Get file name (lFilePath,'Select the library to open','*.lbs') Returns lErrCode
# open the Get File dialog and show only text files
Get file name (lFilePath,'Select the library to open','*.txt;*.doc') Returns lErrCode
```

Get file read-only attribute

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

Get file read-only attribute (*path*, *read-flag*) **Returns** *err-code*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This command returns the current read-only attribute of the file specified in path. If the read-flag parameter returns kTrue the file is read-only, otherwise if kFalse is returned the file is read/write.

It returns an error code (See Error Codes), or zero if no error occurs.

Example

```
# returns the read-only attribute of the omnis.exe in the omnis tree
Calculate lFileName as con(sys(115),'omnis.exe')
Get file read-only attribute (lFileName,lFileAttribute) Returns lErrCode
```

Get files

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

Get files (list-name, first-column, path, file-type [,creator-type] [,8.3]) **Returns** err-code

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This command returns a list of files in a directory or folder.

To list only files of a specified type, specify the file-type, which is a wildcard, such as '*.LBS'.

If you omit the file-type, the command returns the names of all the files.

You specify the list with list-name. The list must have a column defined as list-column-name, where list-column-name is the name of a variable. This column will receive the names of the files found under the specified path-name, including the extension.

On Windows, you can also supply the 8.3 parameter. This defaults to kFalse. If you pass kTrue, then **Get files** returns the 8.3 name equivalent to any long file names.

It returns an error code (See Error Codes), or zero if no error occurs.

When constructing the path to a file or folder, you can use sys(9) to insert the correct path delimiter for the current platform: \ (back-slash) on Windows, or / (forward-slash) for Unix and 64-bit macOS (: colon on 32-bit macOS). In addition, you can use sys(115) to return the full pathname of the folder containing the Omnis executable, including the terminating path separator, which might be useful to reference files in the Omnis tree.

The following example uses **Get files** to build a list of all the libraries in the folder returned by sys(10).

Example

```
Do lFileList.$define(lFileName)
# get the path of the examples folder in the studio tree
Calculate lPathname as con(sys(115),'welcome',sys(9),'examples')
# return the list of all example libraries
Get files (lFileList,lFileName,lPathname,'*.lbs') Returns lErrCode
```

Get folders

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

Get folders (*list-name*, *first-column*, *path* [,*8.3*]) **Returns** err-code

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This command returns a list of folders under the specified path-name.

You specify the list with list-name. The list must have a column defined as list-column-name, where list-column-name is the name of a variable. This column will receive the names of the folders under the specified path-name

On Windows, you can also supply the 8.3 parameter. This defaults to kFalse. If you pass kTrue, then **Get folders** returns the 8.3 name equivalent to any long folder names.

It returns an error code (See Error Codes), or zero if no error occurs.

Example

```
# obtain a list of the folders in the root of your machine
Do lFolderList.$define(lFolderName)
Switch platform()
  Case 'U'
    Get folders (lFolderList,lFolderName,'/')
  Default
    Get folders (lFolderList,lFolderName,'C:\')
End Switch
```

Get statement

Command group	Flag affected	Reversible
SQL Object Commands	NO	NO

Syntax

Get statement *field-name*

Description

This command loads the contents of the SQL statement buffer into a specified field or variable. The field-name parameter can be any Omnis character field or variable. The buffer holds all SQL statements and text entered since the last Begin statement command which have not yet been executed. The square brackets and SQL functions will have been evaluated but the values of indirect @[] square bracket notation will not be available.

Example

```
# Show the sql to the user before creating the table MY_TABLE
Calculate lHostname as con(sys(115),'mydatafile.df1')
Do iSessObj.$logon(lHostname,'','','MYSESSION')
Do iSessObj.$newstatement('MyStatement') Returns lStatObj
Do lRow.$definefromsqlclass('sMySchemaClass')
Do iSessObj.$createnames(lRow) Returns lCreateNames
Begin statement
  Sta: Create Table MY_TABLE ([lCreateNames])
End statement
Get statement lStatment
Yes/No message {Execute [lStatment]}
If flag true
  Do lStatObj.$execdirect()
End If
```

Get text block

Command group	Flag affected	Reversible	Execute on client
Text	NO	NO	YES

Syntax

Get text block *field-name*

Description

This command loads the current contents of the text buffer for the current method stack into the specified field or variable. You build up the text block using the Begin text block and Text: commands. Following an End text block, you can return the contents of the text buffer using the **Get text block** command.

Example

```
Begin text block
  Text: Thought for the day: (Carriage return)
  Text: If a train station is where the train
  Text: stops, what is a work station?
End text block
Get text block lTextString
OK message {[lTextString]}
```

Get working directory

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

Get working directory (*path*) **Returns** *err-code*

Description

Returns the current working directory into path.

Note: The flag is set according to whether Omnis was able to make a call to this external command.

Example

```
# return the current working directory
Get working directory (lDirectory)
```

Go to next selected line

Command group	Flag affected	Reversible	Execute on client
List lines	YES	NO	NO

Syntax

Go to next selected line ([*From start*][,*Backwards*])

Options

From start	If specified, the command starts with the first line of the list rather than the line immediately after the current line
Backwards	If specified, the command steps through the list in reverse order; when used with 'From start' the command starts at the end of the list, otherwise if 'From start' is not specified, it starts with the line before the current line

Deprecated Command

This command has been deprecated and is no longer visible in the Code Assistant in the Code Editor (it will not appear when you type the first few characters), although it is still present in Omnis Studio and will continue to function if used in legacy code. You can show this command by disabling the appropriate Command Filter in the **Modify** menu in the Code Editor.

Description

This command scans a list for selected lines and goes to the first one it finds. It sets the current line (LIST.$line) for the current list (#CLIST) equal to the next selected line in that list.

The **Go to next selected line** command steps through the list starting at the current line (if no options are selected) until a selected line is found. When a selected line is located, LIST.$line is set equal to that line number. If a selected line is not found, the flag is cleared and LIST.$line is unchanged.

The Backwards option causes the list to be searched in descending order; the From start option causes the list to be searched from the start. If both options Backwards and From start are selected, the list is searched from the end.

Example

```
# Transfer the value from line 3 to the 2 selected lines
Set current list lMyList
Define list {lCol1}
For lCount from 1 to 10 step 1
  Add line to list {(lCount)}
End For
Calculate lMyList.$line as 3
Load from list
Select list line(s) {1}
Select list line(s) {5}
Go to next selected line (From start)
Replace line in list
Go to next selected line
Replace line in list
```

Hide docking area

Command group	Flag affected	Reversible	Execute on client
Toolbars	NO	NO	NO

Syntax

Hide docking area {*docking-area* (e.g. kDockingAreaBottom)}

Description

This command closes either the top, bottom, left, or right docking area. The docking area is specified using one of the docking area constants: kDockingAreaTop, kDockingAreaBottom, kDockingAreaLeft, or kDockingAreaRight.

When you close a library, Omnis does not automatically close any docking areas that are open. You must explicitly hide each docking area using Hide docking area. Leaving docking areas open and closing the library containing those docking areas can cause problems in your application.

Example

```
Show docking area {kDockingAreaLeft}
# install toolbar on left docking area
Install toolbar {tbMyToolbar}
# when the library closes, hide the docking area
Hide docking area {kDockingAreaLeft}
# alternatively you can use the following notation
Do $root.$prefs.$dockingareas.$assign(kDockingAreaNone)
```

Hide fields

Command group	Flag affected	Reversible	Execute on client
Fields	NO	YES	NO

Syntax

Hide fields {*list-of-field-names* (Name1,Name2,...)}

Description

This command hides the specified field or list of fields. You can display hidden fields with Show fields.

Example

```
Yes/No message {Do you want to hide fields?}
If flag true
  Begin reversible block
    Hide fields {myField1,myField2}
  End reversible block
End If
# do something
Quit method
# now this method ends and the fields are re-shown as they are in a reversible block
# To hide a single field on the current window
Do $cwind.$objs.myField1.$visible.$assign(kFalse)
# to hide all fields on the current window
Do $cwind.$objs.$sendall($ref.$visible.$assign(kFalse))
```

HTTPClose

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

HTTPClose (*socket*[,*option* {Default zero for complete;1 for partial;2 for abort}]) **Returns** *status*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

HTTPClose 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 **HTTPClose**, to avoid resource leakage.

The most brutal form of **HTTPClose** 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 **HTTPClose** is recommended for use in error handling situations.

The mildest form of **HTTPClose** is a partial close. In this case, the socket is not released, and you will need to call **HTTPClose** 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.

The remaining form of **HTTPClose** is a complete close. In this form, **HTTPClose** 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.

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. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# Connect to the server IP address iHostName on port iPort, send
# the message iMessage and then close the socket
Calculate iHostName as '0.0.0.0'
Calculate iPort as 6000
Calculate lMessage as 'Hello remote application'
HTTPOpen (iHostName,iPort) Returns iSocket
If iSocket>0
  # connected
  HTTPSend (iSocket,lMessage) Returns lCharCount
End If
HTTPClose (iSocket)
```

HTTPGet

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

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

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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

Host 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.

HdrList 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 host, or if none is present, it defaults to 80 or 443, the default port for HTTP or HTTPS respectively (depending on the value of Secure).

Secure is an optional Boolean parameter which indicates if a secure connection is required to the server. Pass kTrue for a secure connection, in which case the built-in security technology will be used, so on Windows 'Secure Channel' (Schannel) is used, on macOS 'Secure Transport' is used, and on Linux OpenSSL is used.

Verify is an optional Boolean parameter which is only significant when Secure is not kFalse. When Verify is kTrue, the command instructs the installed SSL library to verify the server's identity using its certificate; if the verification fails, then the connection will not be established. You can pass Verify as kFalse, to turn off this verification; in this case, the connection will still be encrypted, but there is a chance the server is an impostor. In order to perform the verification, the installed SSL library uses the Certificate Authority Certificates in the cacerts sub-folder of the secure folder in the Omnis folder. If you use your own Certificate Authority to self-sign certificates, you can place its certificate in the cacerts folder, and the installed SSL library will use it after you restart Omnis.

Map+ is an optional Boolean parameter which when true indicates that plus characters in CGI parameter names and values in the CGIList are to be URL encoded as hex.

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. Possible error codes are listed in the Web Command Error Codes Appendix.

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 HdrList.

Example

```
# Open a connection to the web server and read the server response
# into lBuffer
Calculate iHostName as '0.0.0.0'
Do lCGIList.$define(lAttribute,lValue)
Do lCGIList.$add('Name','John Smith')
Do lCGIList.$add('Email','john.smith@smiths.com')
HTTPGet (iHostName,'/default',lCGIList) Returns iSocket
HTTPRead (iSocket,lBuffer) Returns lCharCount
HTTPClose (iSocket) Returns lStatus
```

HTTPHeader

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

HTTPHeader (*socket*,*status*,*headerlist*) **Returns** length

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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 its 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.

Code	Meaning
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. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# When a new connection is received call the method $newconnection
# and reply to the client to confirm the request was completed
HTTPServer ('$newconnection',6001) Returns lStatus
# method $newconnection
Do lHeaderList.$define(lAttribute,lValue)
HTTPHeader (iSocket,200,lHeaderList) Returns lCharCount
```

HTTPMethod

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

HTTPMethod (*socket, uri, method, requesthdrlist, requestcontent, responsestatuscode, responsehdrrow, responsecontent*) **Returns** *status*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

HTTPMethod is a new client command that submits to a Web Server an HTTP request to execute a specified HTTP method.

This Web command requires an existing socket opened with HTTPOpen in order to submit the request. Note that this allows you to sequentially submit more than one request using the same socket connection subject to the rules of HTTP e.g. if the server returns a connection close header in its response, no more requests can be sent on the connection: at this point you need to use HTTPClose

to free the socket resources and open a new connection if you want to send more requests to the server. Re-using a connection like this can be a significant performance improvement, especially when using a secure connection, where the connection set-up time is relatively costly.

Socket is a long integer field containing the socket number of an open HTTP connection.

URI is a Character field containing the URI to which the request is addressed. For example, "/default.html", or "/cgi-bin/mycgiscript". Note that if you wish to send query string parameters you must append them to the URI, using the standard ? syntax, e.g. "/default.html?name=test&value=good". You should encode these parameter names and values using CGIEncode.

Method is a Character field containing the HTTP method to be executed. Note that the method is case-sensitive. Standard HTTP methods such as GET and POST need to be specified in upper case.

RequestHdrList is an Omnis list with two character columns. The list contains the HTTP headers to send to the server. **HTTPMethod** automatically adds a content-length header if you do not specify it in this list, and RequestContent is supplied and not empty.

For example:

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

RequestContent is the content to send to the server with the request. It only makes sense to send content with certain methods, e.g. POST. You can supply either character data, which the command converts to UTF-8 before sending, or binary data.

ResponseStatusCode is an integer field into which Omnis returns the HTTP status code of the HTTP request, e.g. 200 for success.

ResponseHdrRow is a row variable which Omnis populates with the headers received in the response from the server. **HTTPMethod** clears the row and adds columns as necessary. The column name is the header name converted to lower case, with any - characters removed. In addition, if there are headers with the same name,they are combined into a single header with that name,with comma-separated values.

ResponseContent is a binary or character field into which the command stores the content (if any) received in the response from the server. If this is a character field, the command assumes the content is UTF-8 encoded, and converts from UTF-8 to character before storing the response in the field.

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

HTTPOpen

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

HTTPOpen (*hostname*[,*service*|*port*,*secure* {Default kFalse},*verify* {Default kTrue}, *useproxy* {Default kTrue}]) **Returns** *socket*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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 or 443, the default port for HTTP or HTTPS respectively (depending on the value of Secure).

Secure is an optional Boolean parameter which indicates if a secure connection is required to the server. Pass kTrue for a secure connection, in which case the built-in security technology will be used, so on Windows 'Secure Channel' (Schannel) is used, on macOS 'Secure Transport' is used, and on Linux OpenSSL is used.

Verify is an optional Boolean parameter which is only significant when Secure is not kFalse. When Verify is kTrue, the command instructs the installed SSL library to verify the server's identity using its certificate; if the verification fails, then the connection will not be established. You can pass Verify as kFalse, to turn off this verification; in this case, the connection will still be encrypted, but there is a chance the server is an impostor. In order to perform the verification, the installed SSL library uses the Certificate Authority Certificates in the cacerts sub-folder of the secure folder in the Omnis folder. If you use your own Certificate Authority to self-sign certificates, you can place its certificate in the cacerts folder, and the installed SSL library will use it after you restart Omnis.

If *useproxy* is kTrue (the default), and proxy server parameters have been set using HTTPSetProxyServer, **HTTPOpen** connects to the proxy server, setting up a secure tunnel if the proxy server does not have a secure URL, but the requested connection is secure. If kFalse, **HTTPOpen** connects directly to the specified host and port.

If **HTTPOpen** succeeds, socket receives a positive number which is the socket for the new connection to the server. Otherwise, socket receives a negative error code. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# Connect to the server IP address iHostName on port iPort and send
# the message iMessage
Calculate iHostName as '0.0.0.0'
Calculate iPort as 6000
Calculate lMessage as 'Hello remote application'
HTTPOpen (iHostName,iPort) Returns iSocket
If iSocket>0
  # connected
  HTTPSend (iSocket,lMessage) Returns lCharCount
End If
```

HTTPPage

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

HTTPPage (*url*[,*service*|*port*,*verify* {Default kTrue}]) **Returns** *html-text*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

HTTPPage is 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. If you are using a secure connection, the URL must be prefixed with https://.

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, it defaults to 80 or 443, the default port for HTTP or HTTPS respectively.

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. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# Read the html content from lURL into the character variable lHtmlPage
Calculate lUrl as 'http://www.omnis.net/news/index.html'
HTTPPage (lUrl) Returns lHtmlPage
```

HTTPParse

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

HTTPParse (*message*,*headerlist*,*method*,*httpver*[,*uri*,*cgilist*]) **Returns** status

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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.

Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# When a new connection is received call the method $newconnection
# to read and parse the message sent by HTTPPost
HTTPServer ('$newconnection',6001) Returns lStatus
# method $newconnection
HTTPRead (iSocket,lBuffer) Returns lCharCount
Do lHeaderList.$define(lHeaderName,lHeaderValue)
Do lCGIList.$define(lAttribute,lValue)
HTTPParse (lBuffer,lHeaderList,lMethod,lHttpVersion,lUri,lCGIList) Returns lStatus
```

HTTPPost

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

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

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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

Host 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

Attribute	Value
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.

HdrList 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 host, or if none is present, it defaults to 80 or 443, the default port for HTTP or HTTPS respectively (depending on the value of Secure).

Secure is an optional Boolean parameter which indicates if a secure connection is required to the server. Pass kTrue for a secure connection, in which case the built-in security technology will be used, so on Windows 'Secure Channel' (Schannel) is used, on macOS 'Secure Transport' is used, and on Linux OpenSSL is used.

Verify is an optional Boolean parameter which is only significant when Secure is not kFalse. When Verify is kTrue, the command instructs the installed SSL library to verify the server's identity using its certificate; if the verification fails, then the connection will not be established. You can pass Verify as kFalse, to turn off this verification; in this case, the connection will still be encrypted, but there is a chance the server is an impostor. In order to perform the verification, the installed SSL library uses the Certificate Authority Certificates in the cacerts sub-folder of the secure folder in the Omnis folder. If you use your own Certificate Authority to self-sign certificates, you can place its certificate in the cacerts folder, and the installed SSL library will use it after you restart Omnis.

Map+ is an optional Boolean parameter which when true indicates that plus characters in CGI parameter names and values in the CGIList are to be URL encoded as hex.

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. Possible error codes are listed in the Web Command Error Codes Appendix.

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 HdrList.

Example

```
# Post a HTTP request to the server lServer listening on port 6001
Do lCGIList.$define(lAttribute,lValue)
Do lCGIList.$add('Name','John Smith')
Do lCGIList.$add('Email','john.smith@smiths.com')
Calculate lServer as '0.0.0.0.0.0'
HTTPPost (lServer,'\default',lCGIList,lHeaderList,6001) Returns iSocket
```

HTTPRead

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

HTTPRead (*socket*,*buffer*[,*type* {Default zero for server; Non-zero for client}]) **Returns** *received-byte-count*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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 a long integer field containing the socket number of an open HTTP connection.

Buffer is a character or binary field into which **HTTPRead** places the received request or response. If the field is character, then the response must be encoded in UTF-8; in this case, **HTTPRead** converts the received data from UTF-8 to character.

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.

Received-byte-count is a long Integer field which receives the number of bytes placed in Buffer. If an error occurs, an error code less than zero is returned here. Possible error codes are listed in the Web Command Error Codes Appendix.

Note: **HTTPRead** always operates in blocking mode, and will timeout after the connection is inactive for the comms timeout value (which can be changed from its default of 1 minute using the command WebDevSetConfig). 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.

Example

```
# When a new connection is received call the method $newconnection
# to read the message
HTTPServer ('$newconnection',6001) Returns lStatus
# method $newconnection
HTTPRead (iSocket,lBuffer) Returns lByteCount
```

HTTPSend

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

HTTPSend (socket,buffer) **Returns** sent-byte-count

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

Socket is a long integer field containing the socket number of a connected socket.

Buffer is a character or binary field containing the data to send on the socket. If you pass a character field, then **HTTPSend** will convert the data to UTF-8, and then send the UTF-8.

HTTPSend returns the number of bytes it sent to sent-byte-count, a long integer field.

If the socket is in blocking mode, **HTTPSend** always sends all of the data, unless an error occurs.

If the socket is in non-blocking mode, **HTTPSend** sends as much data as it can without blocking.

If an error occurs, **HTTPSend** returns a negative error code

Notes

If the connection to the server is secure, **HTTPSend** always sends the data in blocking mode.

Non-blocking sockets return an error code of -10035 if the socket cannot accept the data to send immediately. Some implementations of socket libraries may have limits on the number of bytes 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 sent-byte-count to determine how much of the buffer has actually been sent; if the value is less than the buffer size, you need to call **HTTPSend** again, to send the rest of the buffer.

It does not make sense to send a character field on a non-blocking socket, because the sent-byte-count corresponds to the sent UTF-8 bytes.

Example

```
# Connect to the server IP address iHostName on port iPort and send
# the message iMessage
Calculate iHostName as '0.0.0.0'
Calculate iPort as 6000
Calculate lMessage as 'Hello remote application'
TCPConnect (iHostName,iPort) Returns iSocket
If iSocket>0
  # connected
  HTTPSend (iSocket,lMessage) Returns lByteCount
End If
```

HTTPServer

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

HTTPServer (*webproc*,*port*[,*workingmessage* {Default non-zero for visible; zero for invisible}]) **Returns** *status*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

HTTPServer invokes a listening socket on a specified port, to receive incoming HTTP requests. This command optionally 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. Possible error codes are listed in the Web Command Error Codes Appendix.

Stopping **HTTPServer**

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

1. Press the Cancel button on the working message displayed by the command.

2. Press the break key sequence (Ctrl-Break/Ctrl-C/Cmnd-period).

3. 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

Example

```
# Listen for incoming http requests on port 6001, call the
# method $newconnection in the current instance when a
# connection arrives.
HTTPServer ('$newconnection',6001) Returns lStatus
```

HTTPSetAuthentication

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

HTTPSetAuthentication (*socket*, *type*, *username*, *password*) **Returns** *status*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

HTTPSetAuthentication provides the parameters needed to authenticate an HTTP request with the server; the command only supports HTTP basic authentication, or no authentication. If you use basic authentication, you are recommended to use a secure connection. Use this command to set up authentication after calling HTTPOpen and before calling HTTPMethod. Note that if you do not want to authenticate the request, a new socket created with HTTPOpen defaults to no authentication, so you do not need to call **HTTPSetAuthentication** in this case.

Socket is a long integer field containing the socket number of an open HTTP connection.

Type is a long integer with value zero for no authentication, or 1 for basic authentication.

Username is a character field containing the user name for basic authentication.

Password is a character field containing the password for basic authentication.

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

HTTPSetProxyServer

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

HTTPSetProxyServer ([*hostname*,*service*|*port*,*secure* {Default kFalse},*verify* {Default kTrue}]) **Returns** *status*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

HTTPSetProxyServer sets the server to which the HTTPGet, HTTPPage and HTTPPost commands connect; the proxy server then requests the URI from the original server (either directly, or via another proxy server). Before **HTTPSetProxyServer** has been called, the commands connect directly to the server for the URI. After setting a proxy server, you can revert to direct connections, by calling **HTTPSetProxyServer** with empty parameters.

Note: There is only a single proxy server setting for the Omnis environment, meaning that it is shared by all threads in the multi-threaded server.

Hostname is a Character field containing the hostname or IP address of the HTTP proxy 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 proxy server. If you specify a service name, the lookup for the port number occurs locally. If you omit this argument, it defaults to 80 or 443, the default port for HTTP or HTTPS respectively (depending on the value of Secure).

Secure is an optional Boolean parameter which indicates if a secure connection is required to the server. Pass kTrue for a secure connection, in which case the built-in security technology will be used, so on Windows 'Secure Channel' (Schannel) is used, on macOS 'Secure Transport' is used, and on Linux OpenSSL is used.

Verify is an optional Boolean parameter which is only significant when Secure is not kFalse. When Verify is kTrue, the command instructs the installed SSL library to verify the server's identity using its certificate; if the verification fails, then the connection will not be established. You can pass Verify as kFalse, to turn off this verification; in this case, the connection will still be encrypted, but there is a chance the server is an impostor. In order to perform the verification, the installed SSL library uses the Certificate Authority Certificates in the cacerts sub-folder of the secure folder in the Omnis folder. If you use your own Certificate Authority to self-sign certificates, you can place its certificate in the cacerts folder, and the installed SSL library will use it after you restart Omnis.

The command returns an integer status, which is less than zero if an error occurs. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# All requests to HTTPGet, HTTPPost and HTTPPage connect to this proxy server
Calculate lHostName as "my.proxy.com"
Calculate lPort as "8080"
HTTPSetProxyServer (lHostName,lPort)
# Clear the proxy server settings, so HTTPGet, HTTPPost and HTTPPage connect directly to the server for the re
HTTPSetProxyServer
```

HTTPSplitHTML

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

HTTPSplitHTML (*message*,*tagtextlist*) **Returns** *status*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# Parse the html from lURL into the list lHtmlTagList
Calculate lUrl as 'http://www.omnis.net/'
HTTPPage (lUrl) Returns lHtmlPage
Do lHtmlTagList.$define(lOpeningHtmlTag,lHtmlText,lClosingHtmlTag)
HTTPSplitHTML (lHtmlPage,lHtmlTagList)
```

HTTPSplitURL

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

HTTPSplitURL (*url*,*hostname*,*uri*) **Returns** *status*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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. If you are using a secure connection, the URL must be prefixed with https://.

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. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# Split lUrl into lHostname and lUri
Calculate lUrl as 'http://www.omnis.net/news/index.html'
HTTPSplitURL (lUrl,lHostName,lUri) Returns lStatus
# lHostName = www.omnis.net, lUri = /news/index.html
```

If calculation

Command group	Flag affected	Reversible	Execute on client
Constructs	NO	NO	YES

Syntax

If calculation

Description

This command tests the result of the calculation and branches if zero. If the result of the calculation is non-zero, the result of the test will be true; a result of zero is interpreted as false. As with all **If** commands, control passes to the next command in the method if the result is true, otherwise to the next End If, Else or Else If in the method.

Example

```
If pSecurityLevel=1
  Open window instance wAministrator
Else
  OK message {This feature is only available to the Administrator}
End If
```

If canceled

Command group	Flag affected	Reversible	Execute on client
Constructs	NO	NO	NO

Syntax

If canceled ([*No refresh*])

Options

No refresh	If specified, the command does not refresh the screen; this may result in improved performance on some platforms, especially when the command is used in each iteration of a loop

Description

This command tests whether the user wishes to cancel execution of the current method, and branches if not. The user requests a cancel by either clicking on a working message Cancel button, or by pressing Ctrl-Break under Windows, Ctrl-C under Linux, or Cmnd-period under macOS. If Enable cancel test at loops is switched on, a loop or other processing may detect a cancel and quit all methods before it is detected by an If canceled command.

Example

```
Calculate #F as 1
Disable cancel test at loops
Working message (Cancel button) {Doing some work}
Repeat
  Redraw working message
  If canceled
```

```
    OK message (Icon,Sound bell ) {Method Terminated.}
    Quit method
  End If
Until flag false
```

If flag false

Command group	Flag affected	Reversible	Execute on client
Constructs	NO	NO	YES

Syntax

If flag false

Description

This command lets you implement a branch or change of processing order within a method depending on the result of the previous command. It tests the flag and if it is false, the commands following the **If flag false** are executed. However, if the flag is true, control branches to the next Else, Else If or End If in the method.

Example

```
# Open the window wMyWindow if it is not already open
Test for window open {wMyWindow}
If flag false
  Open window instance wMyWindow
End If
```

If flag true

Command group	Flag affected	Reversible	Execute on client
Constructs	NO	NO	YES

Syntax

If flag true

Description

This command lets you implement a branch or change of processing order within a method depending on the result of the previous command. It tests the flag and if it is true, the commands following the **If flag true** are executed. However, if the flag is false, control branches to the next Else, Else If or End If in the method.

Example

```
# Test if list line selected sets the flag to true if the line is selected
Set current list iMyList
Test if list line selected {2}
If flag true
  # If the list line is selected, processing continues here.
  OK message {The list line is selected}
End If
```

IMAPCheck

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

IMAPCheck (*socket*[,*stsproc*,*responselist*]) **Returns** status

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

IMAPCheck sends a CHECK command to the IMAP server. The CHECK command requests a checkpoint of the currently selected mailbox. Refer to RFC 3501 for more details.

Before using this command, you must select a mailbox using the IMAPSelectMailbox command

Socket is an Omnis Long Integer field containing a socket opened to an IMAP server using IMAPConnect.

Stsproc is an optional parameter containing the name of an Omnis method that this command calls with status messages. This command calls the method with no parameters, and the status information in the variable #S1. The status information logs protocol messages exchanged on the connection to the server.

Responselist is an optional parameter into which this command places response lines received from the IMAP server. Before calling this command, define the responselist to have a single Character column. When the command returns successfully, the response list contains the untagged and tagged responses received from the IMAP server as a result of executing this command. These sometimes include unsolicited information, for example, an update on the current number of messages in the selected mailbox. Each line in the response list is a response line received from the server. See RFC 3501 for more details, if you need to handle this sort of information.

This command returns an integer, which is less than zero if an error occurred. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
IMAPCheck (iIMAPSocket) Returns lStatus
If lStatus<0
  # The CHECK command failed
End If
```

IMAPConnect

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

IMAPConnect (*server*,*username*,*password*[,*stsproc*,*responselist*,*secure* {Default zero insecure;1 secure;2 use STARTTLS},*verify* {Default kTrue}]) **Returns** *socket*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

IMAPConnect establishes a connection with an IMAP server. The server must support IMAP4rev1. See RFC 3501 for details. If **IMAPConnect** succeeds, it returns the socket opened to the IMAP server. You can use this socket with the other IMAP commands which require a socket argument. If an error occurs, **IMAPConnect** returns an error code, which is less than zero. Possible error codes are listed in the Web Command Error Codes Appendix.

Note that it is essential that you call IMAPDisconnect when you have finished using the connection to the IMAP server.

Server is an Omnis Character field containing the IP address or hostname of an IMAP server. For example: imap.mydomain.com or 255.255.255.254. If the server is not using the default IMAP port (143, or 993 for a secure connection), you can optionally append the port number on which the server is listening, using the syntax server:port, for example imap.mydomain.com:1234.

Username is an Omnis Character field containing the user name that will be used to log in to the IMAP server. The command uses CRAM-MD5 authentication if possible; if CRAM-MD5 is not supported by the server, or fails to authenticate for some reason, the command uses the plain text LOGIN command if the server allows it.

Password is an Omnis character field containing the password for the user specified by the username parameter.

Stsproc is an optional parameter containing the name of an Omnis method that this command calls with status messages. This command calls the method with no parameters, and the status information in the variable #S1. The status information logs protocol messages exchanged on the connection to the server.

Responselist is an optional parameter into which this command places response lines received from the IMAP server. Before calling this command, define the responselist to have a single Character column. When the command returns successfully, the response list contains the untagged and tagged responses received from the IMAP server as a result of executing this command. These sometimes include unsolicited information, for example, an update on the current number of messages in the selected mailbox. Each line in the response list is a response line received from the server. See RFC 3501 for more details, if you need to handle this sort of information. Note you can only omit responselist if it would be the last parameter to be sent, therefore if you include secure and/or verify, then responselist must be included.

Secure is an optional Boolean parameter which indicates if a secure connection is required to the server. Pass kTrue for a secure connection, in which case the built-in security technology will be used, so on Windows 'Secure Channel' (Schannel) is used, on macOS 'Secure Transport' is used, and on Linux OpenSSL is used.

IMAPConnect also supports an alternative secure option, if you pass secure with the value 2, the connection is initially not secure, but after the initial exchange with the server, **IMAPConnect** issues a STARTTLS IMAP command to make the connection secure if the server supports it (see RFC 3501 for details). Authentication occurs after a successful STARTTLS command.

Verify is an optional Boolean parameter which is only significant when Secure is not kFalse. When Verify is kTrue, the command instructs the installed SSL library to verify the server's identity using its certificate; if the verification fails, then the connection will not be established. You can pass Verify as kFalse, to turn off this verification; in this case, the connection will still be encrypted, but there is a chance the server is an impostor. In order to perform the verification, the installed SSL library uses the Certificate Authority Certificates in the cacerts sub-folder of the secure folder in the Omnis folder. If you use your own Certificate Authority to self-sign certificates, you can place its certificate in the cacerts folder, and the installed SSL library will use it after you restart Omnis.

Example

```
# Establish a connection to the IMAP server lServer for user
# lUsername using the password lPassword
Calculate lServer as 'my.imap.server'
Calculate lUserName as 'myusername'
Calculate lPassword as 'mypassword'
IMAPConnect (lServer,lUserName,lPassword) Returns iIMAPSocket
If iIMAPSocket<0
  # Connection failed
End If
```

IMAPCopyMessage

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

IMAPCopyMessage (*socket*,*messageuid*,*destmailboxname*[,*stsproc*,*responselist*]) **Returns** status

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

IMAPCopyMessage copies a message from the currently selected mailbox to another mailbox, using the UID COPY command. Refer to RFC 3501 for more details.

Before using this command, you must select a mailbox using the IMAPSelectMailbox command

Socket is an Omnis Long Integer field containing a socket opened to an IMAP server using IMAPConnect.

Messageuid is an Omnis Long Integer field containing the IMAP Unique Identifier (UID) of the message to be copied.

Destmailboxname is the name of the mailbox into which the message is to be copied.

IMAP mailbox names are left-to-right hierarchical using a single character to separate levels of hierarchy. If you execute IMAPListMailboxes with empty RefName and MailboxName parameters, the returned list has a single line from which you can access the hierarchy separator.

All the Omnis IMAP commands automatically enclose mailbox names in double quotes when sending them to the server.

Stsproc is an optional parameter containing the name of an Omnis method that this command calls with status messages. This command calls the method with no parameters, and the status information in the variable #S1. The status information logs protocol messages exchanged on the connection to the server.

Responselist is an optional parameter into which this command places response lines received from the IMAP server. Before calling this command, define the responselist to have a single Character column. When the command returns successfully, the response list contains the untagged and tagged responses received from the IMAP server as a result of executing this command. These sometimes include unsolicited information, for example, an update on the current number of messages in the selected mailbox. Each line in the response list is a response line received from the server. See RFC 3501 for more details, if you need to handle this sort of information.

This command returns an integer, which is less than zero if an error occurred. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# Copy message with UID 142 from INBOX to sub-folder Test of INBOX
Calculate iMailbox as "INBOX"
IMAPSelectMailbox (iIMAPSocket,iMailbox,iMessages,iRecent,iUIDNext,iUIDValidity,iUnseen) Returns lStatus
If lStatus>=0
  Calculate iMailbox as "INBOX.Test"
  Calculate iUID as 142
  IMAPCopyMessage (iIMAPSocket,iUID,iMailbox) Returns lStatus
  If lStatus<0
    # The copy failed
  End If
End If
```

IMAPCreateMailbox

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

IMAPCreateMailbox (*socket*,*mailboxname*[,*stsproc*,*responselist*]) **Returns** status

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

IMAPCreateMailbox creates a new mailbox on the IMAP server.

Socket is an Omnis Long Integer field containing a socket opened to an IMAP server using IMAPConnect.

Mailboxname is the name of the mailbox to be created.

IMAP mailbox names are left-to-right hierarchical using a single character to separate levels of hierarchy. If you execute IMAPListMailboxes with empty RefName and MailboxName parameters, the returned list has a single line from which you can access the hierarchy separator.

All the Omnis IMAP commands automatically enclose mailbox names in double quotes when sending them to the server.

Stsproc is an optional parameter containing the name of an Omnis method that this command calls with status messages. This command calls the method with no parameters, and the status information in the variable #S1. The status information logs protocol messages exchanged on the connection to the server.

Responselist is an optional parameter into which this command places response lines received from the IMAP server. Before calling this command, define the responselist to have a single Character column. When the command returns successfully, the response list contains the untagged and tagged responses received from the IMAP server as a result of executing this command. These sometimes include unsolicited information, for example, an update on the current number of messages in the selected mailbox. Each line in the response list is a response line received from the server. See RFC 3501 for more details, if you need to handle this sort of information.

This command returns an integer, which is less than zero if an error occurred. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# Create a new folder Test in the INBOX.
# "." is the hierarchy separator
Calculate iMailbox as "INBOX.Test"
IMAPCreateMailbox (iIMAPSocket,iMailbox) Returns lStatus
If lStatus<0
  # The CREATE command failed
End If
```

IMAPDeleteMailbox

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

IMAPDeleteMailbox (socket,mailboxname[,stsproc,responselist]) **Returns** status

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded,allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

IMAPDeleteMailbox deletes a mailbox (and the messages it contains) on the IMAP server.

Socket is an Omnis Long Integer field containing a socket opened to an IMAP server using IMAPConnect.

Mailboxname is the name of the mailbox to be deleted.

IMAP mailbox names are left-to-right hierarchical using a single character to separate levels of hierarchy. If you execute IMAPListMailboxes with empty RefName and MailboxName parameters, the returned list has a single line from which you can access the hierarchy separator.

All the Omnis IMAP commands automatically enclose mailbox names in double quotes when sending them to the server.

Stsproc is an optional parameter containing the name of an Omnis method that this command calls with status messages. This command calls the method with no parameters, and the status information in the variable #S1. The status information logs protocol messages exchanged on the connection to the server.

Responselist is an optional parameter into which this command places response lines received from the IMAP server. Before calling this command, define the responselist to have a single Character column. When the command returns successfully, the response list contains the untagged and tagged responses received from the IMAP server as a result of executing this command. These sometimes include unsolicited information, for example, an update on the current number of messages in the selected mailbox. Each line in the response list is a response line received from the server. See RFC 3501 for more details, if you need to handle this sort of information.

This command returns an integer, which is less than zero if an error occurred. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# Delete folder Test from the INBOX.
# "." is the hierarchy separator
Calculate iMailbox as "INBOX.Test"
IMAPDeleteMailbox (iIMAPSocket,iMailbox) Returns lStatus
If lStatus<0
  # The DELETE command failed
End If
```

IMAPDisconnect

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

IMAPDisconnect (socket[,stsproc,responselist]) **Returns** status

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

IMAPDisconnect closes a connection to an IMAP server.

Socket is an Omnis Long Integer field containing a socket opened to an IMAP server using IMAPConnect.

Stsproc is an optional parameter containing the name of an Omnis method that this command calls with status messages. This command calls the method with no parameters, and the status information in the variable #S1. The status information logs protocol messages exchanged on the connection to the server.

Responselist is an optional parameter into which this command places response lines received from the IMAP server. Before calling this command, define the responselist to have a single Character column. When the command returns successfully, the response list contains the untagged and tagged responses received from the IMAP server as a result of executing this command. These sometimes include unsolicited information, for example, an update on the current number of messages in the selected mailbox. Each line in the response list is a response line received from the server. See RFC 3501 for more details, if you need to handle this sort of information.

This command returns an integer, which is less than zero if an error occurred. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# Close the connection to the IMAP server
IMAPDisconnect (iIMAPSocket)
```

IMAPExpungeMessages

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

IMAPExpungeMessages (*socket*[,*stsproc*,*responselist*]) **Returns** *status*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

IMAPExpungeMessage permanently removes all messages that have the \Deleted flag set from the currently selected mailbox.

Before using this command, you must select a mailbox using the IMAPSelectMailbox command

Socket is an Omnis Long Integer field containing a socket opened to an IMAP server using IMAPConnect.

Stsproc is an optional parameter containing the name of an Omnis method that this command calls with status messages. This command calls the method with no parameters, and the status information in the variable #S1. The status information logs protocol messages exchanged on the connection to the server.

Responselist is an optional parameter into which this command places response lines received from the IMAP server. Before calling this command, define the responselist to have a single Character column. When the command returns successfully, the response list contains the untagged and tagged responses received from the IMAP server as a result of executing this command. These sometimes include unsolicited information, for example, an update on the current number of messages in the selected mailbox. Each line in the response list is a response line received from the server. See RFC 3501 for more details, if you need to handle this sort of information.

This command returns an integer, which is less than zero if an error occurred. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# Delete messages in the selected mailbox with the \Deleted flag
IMAPExpungeMessages (iIMAPSocket) Returns lStatus
If lStatus<0
  # The EXPUNGE command failed
End If
```

IMAPListMailboxes

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

IMAPListMailboxes (socket,refname,mailboxname,list[,stsproc,responselist]) **Returns** status

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

IMAPListMailboxes sends a LIST command to the IMAP server, in order to get a list of a subset of mailbox names from the complete set of all names available to the client.

Socket is an Omnis Long Integer field containing a socket opened to an IMAP server using IMAPConnect.

Refname is an Omnis Character field. The command encloses refname in double quotes, and sends it as the reference name argument of the LIST command. Setting this to an empty string means the mailbox names will be interpreted from the top level. You may also set this as the name of a mailbox, in which case this will be taken as the root of the search, and only mailboxes which are subfolders of this will be included in the search. For full details, see RFC 3501.

Mailboxname is an Omnis Character field. The command encloses mailboxname in double quotes, and sends it as the mailbox name with possible wildcards argument of the LIST command. Setting this to an empty string is a special request, which will return a single list line including the hierarchy separator character. Otherwise it will return a list of mailboxes which match your search criteria. For example, M,*□will return a list of mailboxes beginning with M. For full details, see RFC 3501.

List receives the mailboxes returned by the server. Before calling the command, you must define the list to have seven columns, as follows:

Column	Contains
HasChildren	A long integer which receives the \HasChildren flag value for the mailbox. Not all servers support this flag, and even when a server supports the flag, it may not always supply a value for this flag. Supported values are kFalse if the mailbox has the \HasNoChildren flag, kTrue if the mailbox has the \HasChildren flag, and kUnknown if the mailbox has neither of these flags.
NoInferiors	A long integer which receives the \NoInferiors flag value for the mailbox. kTrue if the mailbox has the \NoInferiors flag, kFalse if not.
NoSelect	A long integer which receives the \NoSelect flag value for the mailbox. kTrue if the mailbox has the \NoSelect flag, kFalse if not.
Marked	A long integer which receives the \Marked flag value for the mailbox. kTrue if the mailbox has the \Marked flag, kFalse if not.
UnMarked	A long integer which receives the \UnMarked flag value for the mailbox. kTrue if the mailbox has the \UnMarked flag, kFalse if not.
Separator	The mailbox hierarchy separator character
MailboxName	The mailbox name

Stsproc is an optional parameter containing the name of an Omnis method that this command calls with status messages. This command calls the method with no parameters, and the status information in the variable #S1. The status information logs protocol messages exchanged on the connection to the server.

Responselist is an optional parameter into which this command places response lines received from the IMAP server. Before calling this command, define the responselist to have a single Character column. When the command returns successfully, the response list contains the untagged and tagged responses received from the IMAP server as a result of executing this command. These sometimes include unsolicited information, for example, an update on the current number of messages in the selected mailbox. Each line in the response list is a response line received from the server. See RFC 3501 for more details, if you need to handle this sort of information.

This command returns an integer, which is less than zero if an error occurred. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# List all mailboxes (folders) in the INBOX (INBOX is a standard IMAP mailbox)
# "." is the hierarchy separator
Do iMailboxList.$define(iHasChildren,iNoInferiors,iNoselect,iMarked,iUnmarked,iSeparator,iMailbox)
Calculate iRefName as "INBOX."
Calculate iMailbox as "%"
IMAPListMailboxes (iIMAPSocket,iRefName,iMailbox,iMailboxList) Returns lStatus
If lStatus<0
  # Command failed
End If
```

IMAPListMessages

Command group	Flag affected	Reversible	Execute on client	P
External commands	YES	NO	NO	A

Syntax

IMAPListMessages (socket,list[,stsproc,responselist]) **Returns** status

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

IMAPListMessages gets the list of messages in the currently selected mailbox.

Before using this command, you must select a mailbox using the IMAPSelectMailbox command

Socket is an Omnis Long Integer field containing a socket opened to an IMAP server using IMAPConnect.

List receives the list of messages in the mailbox. Before calling the command, you must defined the list to have nine columns, as follows:

Column	Contains
UID	A long integer which receives the IMAP Unique Identifier (UID) of the message. Note that the line number in the list is the IMAP message sequence number, at the point the list was generated. It is safest to use UIDs to identify messages.
Size	A long integer which receives the RFC 822 size in bytes of the message.
InternalDate	A date-time which receives the Internal Date of the message. This is typically the date and time that the message was placed in the mailbox.
Answered	A long integer which is set to kTrue if the message has the Answered flag, kFalse if not.
Deleted	A long integer which is set to kTrue if the message has the Deleted flag, kFalse if not.
Draft	A long integer which is set to kTrue if the message has the Draft flag, kFalse if not.
Flagged	A long integer which is set to kTrue if the message has the Flagged flag, kFalse if not.
Recent	A long integer which is set to kTrue if the message has the Recent flag, kFalse if not.
Seen	A long integer which is set to kTrue if the message has the Seen flag, kFalse if not.

Stsproc is an optional parameter containing the name of an Omnis method that this command calls with status messages. This command calls the method with no parameters, and the status information in the variable #S1. The status information logs protocol messages exchanged on the connection to the server.

Responselist is an optional parameter into which this command places response lines received from the IMAP server. Before calling this command, define the responselist to have a single Character column. When the command returns successfully, the response list contains the untagged and tagged responses received from the IMAP server as a result of executing this command. These sometimes

include unsolicited information, for example, an update on the current number of messages in the selected mailbox. Each line in the response list is a response line received from the server. See RFC 3501 for more details, if you need to handle this sort of information.

This command returns an integer, which is less than zero if an error occurred. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# List all messages in the currently selected mailbox
Do iMessageList.$define(iUID,iSize,iInternalDate,iAnswered,iDeleted,iDraft,iFlagged,iRecent,iSeen)
IMAPListMessages (iIMAPSocket,iMessageList) Returns lStatus
If lStatus<0
  # Command failed
End If
```

IMAPListSubscribedMailboxes

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

IMAPListSubscribedMailboxes (socket,refname,mailboxname,list[,stsproc,responselist]) **Returns** status

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

IMAPListSubscribedMailboxes sends an LSUB command to the IMAP server, in order to get a list of a subset of mailbox names from the complete set of all subscribed names available to the client.

Socket is an Omnis Long Integer field containing a socket opened to an IMAP server using IMAPConnect.

Refname is an Omnis Character field. The command encloses refname in double quotes, and sends it as the reference name argument of the LIST command. Setting this to an empty string means the mailbox names will be interpreted from the top level. You may also set this as the name of a mailbox, in which case this will be taken as the root of the search, and only mailboxes which are subfolders of this will be included in the search. For full details, see RFC 3501.

Mailboxname is an Omnis Character field. The command encloses mailboxname in double quotes, and sends it as the mailbox name with possible wildcards argument of the LIST command. Setting this to an empty string is a special request, which will return a single list line including the hierarchy separator character. Otherwise it will return a list of mailboxes which match your search criteria. For example, M¦*□will return a list of mailboxes beginning with M. For full details, see RFC 3501.

List receives the mailboxes returned by the server. Before calling the command, you must define the list to have seven columns, as follows:

Column	Contains
HasChildren	A long integer which receives the \HasChildren flag value for the mailbox. Not all servers support this flag, and even when a server supports the flag, it may not always supply a value for this flag. Supported values are kFalse if the mailbox has the \HasNoChildren flag, kTrue if the mailbox has the \HasChildren flag, and kUnknown if the mailbox has neither of these flags.
NoInferiors	A long integer which receives the \NoInferiors flag value for the mailbox. kTrue if the mailbox has the \NoInferiors flag, kFalse if not.
NoSelect	A long integer which receives the \NoSelect flag value for the mailbox. kTrue if the mailbox has the \NoSelect flag, kFalse if not.

Column	Contains
Marked	A long integer which receives the \Marked flag value for the mailbox. kTrue if the mailbox has the \Marked flag, kFalse if not.
UnMarked	A long integer which receives the \UnMarked flag value for the mailbox. kTrue if the mailbox has the \UnMarked flag, kFalse if not.
Separator	The mailbox hierarchy separator character
MailboxName	The mailbox name

Stsproc is an optional parameter containing the name of an Omnis method that this command calls with status messages. This command calls the method with no parameters, and the status information in the variable #S1. The status information logs protocol messages exchanged on the connection to the server.

Responselist is an optional parameter into which this command places response lines received from the IMAP server. Before calling this command, define the responselist to have a single Character column. When the command returns successfully, the response list contains the untagged and tagged responses received from the IMAP server as a result of executing this command. These sometimes include unsolicited information, for example, an update on the current number of messages in the selected mailbox. Each line in the response list is a response line received from the server. See RFC 3501 for more details, if you need to handle this sort of information.

This command returns an integer, which is less than zero if an error occurred. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# List all subscribed mailboxes (folders) in the INBOX (INBOX is a standard IMAP mailbox)
# "." is the hierarchy separator
Do iMailboxList.$define(iHasChildren,iNoInferiors,iNoselect,iMarked,iUnmarked,iSeparator,iMailbox)
Calculate iRefName as "INBOX."
Calculate iMailbox as "%"
IMAPListSubscribedMailboxes (iIMAPSocket,iRefName,iMailbox,iMailboxList) Returns lStatus
If lStatus<0
  # Command failed
End If
```

IMAPNoOp

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

IMAPNoOp (socket[,stsproc,responselist]) **Returns** status

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded,allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

IMAPNoOp sends a NOOP command to the IMAP server. The command itself does nothing, but clients can use the NOOP command to poll the server to get status updates via untagged responses (which will be placed in the responselist parameter if it is present). See RFC 3501 for details. Note that the IMAPListMessages command automatically sends a NOOP command before fetching the list of messages; this ensures that new messages are returned in the list.

Socket is an Omnis Long Integer field containing a socket opened to an IMAP server using IMAPConnect.

Stsproc is an optional parameter containing the name of an Omnis method that this command calls with status messages. This command calls the method with no parameters, and the status information in the variable #S1. The status information logs protocol messages exchanged on the connection to the server.

Responselist is an optional parameter into which this command places response lines received from the IMAP server. Before calling this command, define the responselist to have a single Character column. When the command returns successfully, the response list contains the untagged and tagged responses received from the IMAP server as a result of executing this command. These sometimes include unsolicited information, for example, an update on the current number of messages in the selected mailbox. Each line in the response list is a response line received from the server. See RFC 3501 for more details, if you need to handle this sort of information.

This command returns an integer, which is less than zero if an error occurred. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# Issue a NOOP command to poll the server
Do iResponseList.$define(iResponse)
IMAPNoOp (iIMAPSocket,"",iResponseList) Returns lStatus
If lStatus<0
  # NOOP command failed
End If
```

IMAPRecvHeaders

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

IMAPRecvHeaders (socket,messageuid,headers[,stsproc,responselist]) **Returns** status

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded,allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

IMAPRecvHeaders receives the headers for a specified message in the currently selected mailbox. The received headers are in RFC 822 format. You can pass the received headers to the MailSplit command, in order to parse them.

Before using this command, you must select a mailbox using the IMAPSelectMailbox command

Socket is an Omnis Long Integer field containing a socket opened to an IMAP server using IMAPConnect.

Messageuid is an Omnis Long Integer field containing the IMAP Unique Identifier (UID) of the message for which the headers are to be retrieved.

Headers is an Omnis Binary or Character field which receives the RFC 822 headers for the message. For correct results with many of the encodings supported by MailSplit you must receive into a Binary field.

Stsproc is an optional parameter containing the name of an Omnis method that this command calls with status messages. This command calls the method with no parameters, and the status information in the variable #S1. The status information logs protocol messages exchanged on the connection to the server.

Responselist is an optional parameter into which this command places response lines received from the IMAP server. Before calling this command, define the responselist to have a single Character column. When the command returns successfully, the response list contains the untagged and tagged responses received from the IMAP server as a result of executing this command. These sometimes include unsolicited information, for example, an update on the current number of messages in the selected mailbox. Each line in the response list is a response line received from the server. See RFC 3501 for more details, if you need to handle this sort of information.

This command returns an integer, which is less than zero if an error occurred. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# Receive headers for message with UID 142 in the currently selected mailbox
Calculate iUID as 142
IMAPRecvHeaders (iIMAPSocket,iUID,lHeaders) Returns lStatus
If lStatus<0
  # Command failed
End If
```

IMAPRecvMessage

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

IMAPRecvMessage (socket,messageuid,message[,stsproc,responselist]) **Returns** status

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

IMAPRecvMessage receives a specified message in the currently selected mailbox. The received message is in RFC 822 format. You can pass the received message to the MailSplit command, in order to parse it.

Before using this command, you must select a mailbox using the IMAPSelectMailbox command

Socket is an Omnis Long Integer field containing a socket opened to an IMAP server using IMAPConnect.

Messageuid is an Omnis Long Integer field containing the IMAP Unique Identifier (UID) of the message to be retrieved.

Message is an Omnis Binary or Character field which receives the RFC 822 format message. For correct results with many of the encodings supported by MailSplit you must receive into a Binary field.

Stsproc is an optional parameter containing the name of an Omnis method that this command calls with status messages. This command calls the method with no parameters, and the status information in the variable #S1. The status information logs protocol messages exchanged on the connection to the server.

Responselist is an optional parameter into which this command places response lines received from the IMAP server. Before calling this command, define the responselist to have a single Character column. When the command returns successfully, the response list contains the untagged and tagged responses received from the IMAP server as a result of executing this command. These sometimes include unsolicited information, for example, an update on the current number of messages in the selected mailbox. Each line in the response list is a response line received from the server. See RFC 3501 for more details, if you need to handle this sort of information.

This command returns an integer, which is less than zero if an error occurred. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# Receive message with UID 142 in the currently selected mailbox
Calculate iUID as 142
IMAPRecvMessage (iIMAPSocket,iUID,lMessage) Returns lStatus
If lStatus<0
  # Command failed
End If
```

IMAPRenameMailbox

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

IMAPRenameMailbox (socket,oldmailboxname,newmailboxname[,stsproc,responselist]) **Returns** status

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

IMAPRenameMailbox renames a mailbox.

Socket is an Omnis Long Integer field containing a socket opened to an IMAP server using IMAPConnect.

Oldmailboxname is the name of the mailbox to be renamed.

Newmailboxname is the new name for the mailbox.

Stsproc is an optional parameter containing the name of an Omnis method that this command calls with status messages. This command calls the method with no parameters, and the status information in the variable #S1. The status information logs protocol messages exchanged on the connection to the server.

Responselist is an optional parameter into which this command places response lines received from the IMAP server. Before calling this command, define the responselist to have a single Character column. When the command returns successfully, the response list contains the untagged and tagged responses received from the IMAP server as a result of executing this command. These sometimes include unsolicited information, for example, an update on the current number of messages in the selected mailbox. Each line in the response list is a response line received from the server. See RFC 3501 for more details, if you need to handle this sort of information.

This command returns an integer, which is less than zero if an error occurred. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# Rename folder Test in the INBOX to Test2
# "." is the hierarchy separator
Calculate iMailbox as "INBOX.Test"
Calculate iNewMailbox as "INBOX.Test2"
IMAPRenameMailbox (iIMAPSocket,iMailbox,iNewMailbox) Returns lStatus
If lStatus<0
  # RENAME command failed
End If
```

IMAPSelectMailbox

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

IMAPSelectMailbox (socket,mailboxname,messages,recent,uidnext,uidvalidity,unseen[,stsproc,responselist]) **Returns** status

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

IMAPSelectMailbox makes a mailbox the currently selected mailbox. Certain IMAP commands operate in the context of a selected mailbox, meaning that this command needs to be executed first.

Socket is an Omnis Long Integer field containing a socket opened to an IMAP server using IMAPConnect.

Mailboxname is the name of the mailbox to be selected.

IMAP mailbox names are left-to-right hierarchical using a single character to separate levels of hierarchy. If you execute IMAPListMailboxes with empty RefName and MailboxName parameters, the returned list has a single line from which you can access the hierarchy separator.

All the Omnis IMAP commands automatically enclose mailbox names in double quotes when sending them to the server.

Messages is an Omnis Long Integer field which receives the number of messages in the selected mailbox, if the command succeeds. If the count is not received in the response to the IMAP SELECT command, this value can be zero.

Recent is an Omnis Long Integer field which receives the number of messages in the selected mailbox with the \Recent flag set, if the command succeeds.If the count is not received in the response to the IMAP SELECT command, this value can be zero.

Uidnext is an Omnis Long Integer field which receives the next unique identifier value for the selected mailbox, if the command succeeds.If the value is not received in the response to the IMAP SELECT command, this value can be zero.

Uidvalidity is an Omnis Long Integer field which receives the unique identifier validity value for the selected mailbox, if the command succeeds.If the value is not received in the response to the IMAP SELECT command, this value can be zero.

Unseen is an Omnis Long Integer field which receives the message sequence number of the first unseen message in the selected mailbox, if the command succeeds.If the value is not received in the response to the IMAP SELECT command, this value can be zero.

Stsproc is an optional parameter containing the name of an Omnis method that this command calls with status messages. This command calls the method with no parameters, and the status information in the variable #S1. The status information logs protocol messages exchanged on the connection to the server.

Responselist is an optional parameter into which this command places response lines received from the IMAP server. Before calling this command, define the responselist to have a single Character column. When the command returns successfully, the response list contains the untagged and tagged responses received from the IMAP server as a result of executing this command. These sometimes include unsolicited information, for example, an update on the current number of messages in the selected mailbox. Each line in the response list is a response line received from the server. See RFC 3501 for more details, if you need to handle this sort of information.

This command returns an integer, which is less than zero if an error occurred. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# Make INBOX the currently selected mailbox
Calculate iMailbox as "INBOX"
IMAPSelectMailbox (iIMAPSocket,iMailbox,iMessages,iRecent,iUIDNext,iUIDValidity,iUnseen) Returns lStatus
If lStatus<0
  # SELECT command failed
End If
```

IMAPSetMessageFlags

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

IMAPSetMessageFlags (socket,messageuid,answered,deleted,draft,flagged,seen[,stsproc,responselist]) **Returns** status

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

IMAPSetMessageFlags adds and removes flags for a message in the currently selected mailbox. Each flag value can be passed as follows:

Value	Meaning
kFalse	Remove the flag from the message.
kTrue	Add the flag to the message.
kUnknown	Leave the flag unchanged.

Before using this command, you must select a mailbox using the IMAPSelectMailbox command

Socket is an Omnis Long Integer field containing a socket opened to an IMAP server using IMAPConnect.

Messageuid is an Omnis Long Integer field containing the IMAP Unique Identifier (UID) of the message for which the flags are to be set.

Answered is the flag value (as defined above) for \Answered.

Deleted is the flag value (as defined above) for \Deleted.

Draft is the flag value (as defined above) for \Draft.

Flagged is the flag value (as defined above) for \Flagged.

Seen is the flag value (as defined above) for \Seen.

Stsproc is an optional parameter containing the name of an Omnis method that this command calls with status messages. This command calls the method with no parameters, and the status information in the variable #S1. The status information logs protocol messages exchanged on the connection to the server.

Responselist is an optional parameter into which this command places response lines received from the IMAP server. Before calling this command, define the responselist to have a single Character column. When the command returns successfully, the response list contains the untagged and tagged responses received from the IMAP server as a result of executing this command. These sometimes include unsolicited information, for example, an update on the current number of messages in the selected mailbox. Each line in the response list is a response line received from the server. See RFC 3501 for more details, if you need to handle this sort of information.

This command returns an integer, which is less than zero if an error occurred. Possible error codes are listed in the Web Command Error Codes Appendix.

Note:

You use **IMAPSetMessageFlags** to delete a message, by adding the \Deleted flag to the message. You can then permanently delete all messages in the currently selected mailbox with the \Deleted flag set, by calling IMAPExpungeMessages

Example

```
# Mark message 142 in the currently selected mailbox as deleted
Calculate iUID as 142
Calculate iAnswered as kUnknown
Calculate iDeleted as kTrue
Calculate iDraft as kUnknown
Calculate iFlagged as kUnknown
Calculate iSeen as kUnknown
IMAPSetMessageFlags (iIMAPSocket,iUID,iAnswered,iDeleted,iDraft,iFlagged,iSeen) Returns lStatus
If lStatus<0
  # Command failed
End If
```

IMAPSubscribeMailbox

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

IMAPSubscribeMailbox (socket,mailboxname[,stsproc,responselist]) **Returns** status

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

IMAPSubscribeMailbox issues a SUBSCRIBE command to the server, to add a specified mailbox name to the server's set of "active" or "subscribed" mailboxes as returned by IMAPListSubscribedMailboxes.

Socket is an Omnis Long Integer field containing a socket opened to an IMAP server using IMAPConnect.

Mailboxname is the name of the mailbox.

IMAP mailbox names are left-to-right hierarchical using a single character to separate levels of hierarchy. If you execute IMAPListMailboxes with empty RefName and MailboxName parameters, the returned list has a single line from which you can access the hierarchy separator.

All the Omnis IMAP commands automatically enclose mailbox names in double quotes when sending them to the server.

Stsproc is an optional parameter containing the name of an Omnis method that this command calls with status messages. This command calls the method with no parameters, and the status information in the variable #S1. The status information logs protocol messages exchanged on the connection to the server.

Responselist is an optional parameter into which this command places response lines received from the IMAP server. Before calling this command, define the responselist to have a single Character column. When the command returns successfully, the response list contains the untagged and tagged responses received from the IMAP server as a result of executing this command. These sometimes include unsolicited information, for example, an update on the current number of messages in the selected mailbox. Each line in the response list is a response line received from the server. See RFC 3501 for more details, if you need to handle this sort of information.

This command returns an integer, which is less than zero if an error occurred. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# Add INBOX.Test to the subscribed mailboxes
# "." is the hierarchy separator
Calculate iMailbox as "INBOX.Test"
IMAPSubscribeMailbox (iIMAPSocket,iMailbox) Returns lStatus
If lStatus<0
  # SUBSCRIBE command failed
End If
```

IMAPUnsubscribeMailbox

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

IMAPUnsubscribeMailbox (socket,mailboxname[,stsproc,responselist]) **Returns** status

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

IMAPUnsubscribeMailbox issues an UNSUBSCRIBE command to the server, to remove a specified mailbox name from the server's set of "active" or "subscribed" mailboxes as returned by IMAPListSubscribedMailboxes.

Socket is an Omnis Long Integer field containing a socket opened to an IMAP server using IMAPConnect.

Mailboxname is the name of the mailbox.

IMAP mailbox names are left-to-right hierarchical using a single character to separate levels of hierarchy. If you execute IMAPListMailboxes with empty RefName and MailboxName parameters, the returned list has a single line from which you can access the hierarchy separator.

All the Omnis IMAP commands automatically enclose mailbox names in double quotes when sending them to the server.

Stsproc is an optional parameter containing the name of an Omnis method that this command calls with status messages. This command calls the method with no parameters, and the status information in the variable #S1. The status information logs protocol messages exchanged on the connection to the server.

Responselist is an optional parameter into which this command places response lines received from the IMAP server. Before calling this command, define the responselist to have a single Character column. When the command returns successfully, the response list contains the untagged and tagged responses received from the IMAP server as a result of executing this command. These sometimes include unsolicited information, for example, an update on the current number of messages in the selected mailbox. Each line in the response list is a response line received from the server. See RFC 3501 for more details, if you need to handle this sort of information.

This command returns an integer, which is less than zero if an error occurred. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# Remove INBOX.Test from the subscribed mailboxes
# "." is the hierarchy separator
Calculate iMailbox as "INBOX.Test"
IMAPUnsubscribeMailbox (iIMAPSocket,iMailbox) Returns lStatus
If lStatus<0
  # UNSUBSCRIBE command failed
End If
```

Import data

Command group	Flag affected	Reversi
Importing and Exporting	YES	NO

Syntax

Import data list-or-row-name

Description

This command reads the next data item into the specified list or row variable. You use the **Import data** command to import data from a file or port. Once you select an import file or port, and issue a Prepare for import command, Import data adds the data to the specified list or row variable.

If a record is successfully read from the file or port, Omnis sets the flag. An error occurs if the import file or port is closed or if the specified list or row variable does not exist. The flag is set after reading a record successfully.

After the import is complete, you should follow **Import data** with an End import and the appropriate Close import file or Close port.

There is a one-to-one mapping between the columns or fields in the import file and the columns in the list or row variable. Therefore, if there are fewer columns or fields in the import file than in the list or row, the excess import columns or fields are ignored. Likewise, if there are more columns in the list or row than in the import file, the excess columns are left blank.

The 'LFonlyLineTermination' item in the 'default' section of config.json allows you to control how carriage returns and line feeds are handled when importing data from a file. If true, when Omnis imports a tab- or comma-separated file and the file has no Carriage Return (CR) line separators, Omnis will then check for Line Feed (LF) line separators and use these to break record rows.

Example

```
# import from a csv file called myImport.txt in the root of your omnis tree
Calculate lImportPath as con(sys(115),'myImport.txt')
Set import file name {[lImportPath]}
Prepare for import from file {Delimited (commas)}
Import data lImportList
End import
Close import file
```

Import field from file

Command group	Flag affected	Reversi
Importing and Exporting	YES	NO

Syntax

Import field from file into field-name ([Single character][,Leave in buffer])

Options

Single character	If specified, the command reads a single character at a time
Leave in buffer	If specified, the command leaves the data it returns in the buffer meaning that the next call to the command will return the same value

Description

This command reads a line of characters from the current import file to the specified field. It lets you read fields from a file without using a window and Import data. Usually the command reads a whole line at a time but there are options which modify this.

The Single character option tells Omnis to read a single character at a time. If the field is a Character or a National field, it is set to have a length of one, containing the single character imported from the file. If the field is a Number field, the field value is set to the ASCII code of the single character imported from the file.

The Leave in buffer option tells Omnis to read the string or single character but not remove it from the buffer. Therefore, the next **Import field from file** will read exactly the same value.

An error will occur if the import file has not been opened; Omnis clears the flag on reaching the end of the file. Do not mix Import data and **Import field from file** because they use the input buffer in different ways.

Example

```
# import from a csv file called myImport.txt in the root of your omnis tree
Calculate lImportPath as con(sys(115),'myImport.txt')
Set import file name {[lImportPath]}
Prepare for import from file {Delimited (commas)}
Repeat
  Import field from file int lImportField
Until lImportField='start data'
Do method ImportData
Close import file
```

Import field from port

Command group	Flag affected	Reversi
Importing and Exporting	YES	NO

Syntax

Import field from port into field-name ([Single character][,Leave in buffer][,Clear buffer][,Do not wait])

Options

Single character	If specified the command reads a single character at a time
Leave in buffer	If specified, the command leaves the data it returns in the buffer meaning that the next call to the command will return the same value
Clear buffer	If specified, the command clears the import buffer before executing
Do not wait	If specified, the command will not wait until data is available

Description

This command reads a line of characters from the current port to the specified field. **Import field from port** lets you read fields from a port without using a window and Import data. Usually the command reads a whole line at a time but there are options which modify this:

Single character tells Omnis to read a single character at a time. If the field is a Character or a National field, it is set to have a length of one, containing the single character imported from the port. If the field is a Number field, the field value is set to the ASCII code of the single character imported from the port.

Leave in buffer tells Omnis to read the string or single character but not remove it from the buffer. Therefore, the next **Import field from port** command will read exactly the same value.

Clear buffer clears the import buffer so that previously received values are ignored.

Do not wait prevents Omnis from waiting until a string or character is available.

An error will occur if the import port has not been opened; Omnis clears the flag if nothing has been read. Do not mix the Import data and **Import field from port** commands because they use the input buffer in different ways.

Example

```
Set port name {COM1:}
Prepare for import from port {One field per line}
Repeat
  Import field from port int lImportField
Until lImportField='start data'
Do method ImportData
Close import file
```

Import fields

Command group	Flag affected	Reversi
Importing and Exporting	YES	NO

Syntax

Import fields (Insert records|Update records[,Indirect][,Disable messages]) {list-of-field-names (Name1,Name2,...)}

Types

Insert records	The command inserts new records
Update records	The command searches for existing records in the file and updates the records that it finds; data for which there is no matching record is ignored

Options

Indirect	If specified, the command uses the contents of the first field as the list of fields
Disable messages	If specified, the command does not open messages requiring a user response and instead it writes a limited amount of information to the trace log

Description

Import fields imports the data for the list of fields from the current import file into the data file. It provides runtime access to the functionality of the import data dialog in the IDE. The command sets the main file for the import to the file corresponding to the first field in the list.

The Insert records option causes the command to insert new records for the data in the file being imported.

The Update records option causes the command to search for an existing record in the data file, for each record in the file being imported, and then update that record. Import records for which there is no matching record in the data file are ignored.

Example

```
# import from a csv file called myImport.txt in the root of your omnis tree
Calculate lImportPath as con(sys(115),'myImport.txt')
Set import file name {[lImportPath]}
Prepare for import from file {Delimited (commas)}
Import fields (Insert records) {fCustomers.Surname,fCustomers.FirstName}
End import
Close import file
```

Insert line in list

Command group	Flag affected	Reversible	Execute on client
List lines	YES	NO	NO

Syntax

Insert line in list {*line-number* (*values*) {default is current line}}

Deprecated Command

This command has been deprecated and is no longer visible in the Code Assistant in the Code Editor (it will not appear when you type the first few characters), although it is still present in Omnis Studio and will continue to function if used in legacy code. You can show this command by disabling the appropriate Command Filter in the **Modify** menu in the Code Editor.

Description

This command takes the current field values and inserts them at a particular line in the list. The new line is inserted before the specified line and all the lines below the specified line are moved down one place.

If a set of comma-separated values is included as a parameter, these values are read (in order) into the columns of the new line. In this case, the field names for the columns are not used to specify the data for the new line.

You can specify the line number using a calculation. However, if the parameter for the command is empty or evaluates to zero, the current line is used, that is, the field values are inserted at the current line and all other lines are moved down one place.

If there is no current line (*LIST.$line* = 0), the field values are added at the end of the list. If the line is beyond the current end of the list (for example, the LIST.$line given is greater than *LIST.$linecount*),**Insert line in list** is equivalent to *Add line to list*. The flag is cleared if the list is already at its maximum size (*LIST.$linemax*).

Example

```
# Insert 10 lines in between the 2 exisiting lines
Set current list lMyList
Define list {lName,lAge}
Insert line in list {('Fred',10)}
Insert line in list {('George',20)}
For lCount from 1 to 10 step 1
  Insert line in list {2 ('Harry',22)}
End For
# Alternatively, you can use the $addbefore() and $addafter() methods to add lines to a list
Do lMyList.$addbefore(1,'Harry',22)
Do lMyList.$addafter(2,'William',31)
```

Install menu

Command group	Flag affected	Reversible	Execute on client
Menus	YES	YES	NO

Syntax

Install menu class-name[/instance-name] [(parameters)]

Description

This command installs an instance of the specified menu class on the main menu bar and assigns an instance name. The default instance name is the name of the menu class. The flag is set if the menu is installed.

You can choose the menu class from a list containing your own menus in the current library, and the standard menus *File, *Edit, and so on. When the menu instance is installed its $construct() method is called receiving any parameters passed.

If you use the **Install menu** command in a reversible block, the menu instance is removed from the menu bar when the method terminates. However, the order of the menus on the menu bar may not necessarily be the same as before.

Example

```
# Install the menu mView and pass the parameter
# lView to its $construct method
Calculate lView as 'Large'
Install menu mView (lView)
# mView $constuct method
Do $cinst.$objs.[pView].$checked.$assign(kTrue) ## Check the menu line pView
# Alternatively, you can install a menu using $open
Do $clib.$menus.mView.$open()
```

Install toolbar

Command group	Flag affected	Reversible	Execute on client
Toolbars	NO	NO	NO

Syntax

Install toolbar {*class*[/*instance*][/*dock-area*/*l*/*t*][(*params*)]{defaults are class settings}}

Description

This command installs the specified toolbar class into the named docking area. You specify the docking area using one of the toolbar constants: kDockingAreaTop, kDockingAreaBottom, kDockingAreaLeft,kDockingAreaRight, or kDockingAreaFloating. If you omit the docking area name the toolgroup is installed into the docking area specified in the class. You can install multiple toolbars onto the same docking area.

If the specified docking area is kDockingAreaFloating, then you can specify the left (/l) and top (/t) position of the toolbar instance in pixels .

Example

```
# show the left and right toolbar docking areas
Show docking area {kDockingAreaLeft}
Show docking area {kDockingAreaRight}
# install a toolbar into each docking area
Install toolbar {tbMyToolbar/kDockingAreaLeft}
Install toolbar {tbMyOtherToolbar/kDockingAreaRight}
# or you can install a toolbar notationally
Do $clib.$toolbars.tbMyToolbar.$open('*',kDockingAreaLeft) Returns lToolBarRef
```

Invert selection for line(s)

Command group	Flag affected	Reversible	Execute on client
List lines	YES	NO	NO

Syntax

Invert selection for line(s) ([*All lines*]) {*line-number* (*calculation*)}

Options

All lines	If specified, the command affects all the lines in the list

Deprecated Command

This command has been deprecated and is no longer visible in the Code Assistant in the Code Editor (it will not appear when you type the first few characters), although it is still present in Omnis Studio and will continue to function if used in legacy code. You can show this command by disabling the appropriate Command Filter in the **Modify** menu in the Code Editor.

Description

This command inverts the selection state of a line, that is, from selected to deselected or vice-versa. You can specify a particular line in the list by entering either a number or a calculation. You can show the selection state on the window by invoking the Redraw lists (Selection only) command.

The All lines option inverts the selection states of all lines of the current list. If no line number is given, the current line selection is inverted. When a list is saved in the data file, the selection state of each line is stored. The following example selects all but the middle line of the list:

Example

```
# Select list lines 2 and 4 and then invert the selection
# so list lines 1,3 and 5 are selected
Set current list lMyList
Define list {lName,lAge}
Add line to list {('Fred',10)}
Add line to list {('George',20)}
Add line to list {('Harry',22)}
Add line to list {('William',31)}
Add line to list {('David',62)}
Select list line(s) {2}
Select list line(s) {4}
Invert selection for line(s) (All lines)
```

JavaScript:

Command group	Flag affected	Reversible	Execute on c
Calculations	NO	NO	YES

Syntax

JavaScript: javascript-code

Description

Use this command to insert raw JavaScript code into the method in the client methods JavaScript file. Consequently, this command cannot be run in a server method.

A JavaScript editor will pop up when you enter or edit a line of JavaScript code. You can also paste in a block of JavaScript code from the clipboard.

You cannot insert an inline comment on any lines in a JavaScript: code block.

```
Javascript: alert("I am an alert box!");
```

Jump to start of loop

Command group	Flag affected	Reversible	Execute on client
Constructs	NO	NO	YES

Syntax

Jump to start of loop

Description

This command jumps to the Until or While command at the beginning of the current loop, missing out all commands after the jump. When used in a While–End While loop, **Jump to start of loop** jumps to the start of the loop so that Omnis can make the While test; the loop continues or terminates depending on the result of this test, whereas, Break to end of loop automatically terminates the loop regardless of the value of the condition. Placing a Jump outside a loop causes an error.

Example

```
# Only calculate lBalance if an account number has been entered
Calculate lBalance as 0
Repeat

  Prompt for input Account Number Returns lAccountNumber (Cancel button)
  If flag false ## cancel button
    Break to end of loop
  Else If len(lAccountNumber)=0 ## no account number entered
    OK message {Please enter an account number}
    Jump to start of loop

  End If
  Calculate lBalance as 100

Until lBalance>0
```

Launch program

Command group	Flag affected	Reversible
Operating system	YES	NO

Syntax

Launch program program-name|program-name,document-name **Returns** return-value ([Do not quit Omnis])

Options

Do not quit Omnis	This option is ignored on platforms other than macOS. When running on macOS, specify this option to prevent Omnis from closing after launching the program

Description

This command launches the specified program.

On Windows and Linux, this command behaves just like the command Start program normal, except that you can also wait for the output from the program (see below). The *Do not quit Omnis* option is ignored.

On Windows and Linux, you can run a command line program, and receive the output from the program via the Returns clause of the command. If a variable is specified in the Returns clause, Omnis Studio waits for the executable to terminate before continuing, and returns the output from the command in the variable.

On Windows, you can omit the program name, and supply just the document name prefixed by a comma. This will open the document in the application associated with its file extension.

The rest of this command description applies only to macOS. If you include a file name, the application is launched with the file name as a document. If the specified file name represents a document which the program cannot understand, it will be ignored. You must specify pathnames for the program and document, as shown in the example below.

You can reference either the application (with the .app suffix) or the executable in the bundle. For example, to launch iTunes you can specify either :

/Applications/iTunes.app
or
/Applications/iTunes.app/Contents/macOS/iTunes

The default action is to quit Omnis, but the *Do not quit Omnis* option lets you keep Omnis open. If you choose this option, Omnis will continue to run in the background, concurrently with the new program. A new program launched by Omnis will always be opened on top, even if Omnis is already in the background. The flag is set false if an error is detected, for example, if a program or file name cannot be found. When you execute **Launch program**, control passes from your application to the operating system and there is no automatic way of returning to Omnis.

Example

```
# Launch the specified program
Launch program c:\windows\notepad.exe
If flag false
  OK message (Icon,Sound bell ) {Couldn't find notepad.exe}
End If
```

Optionally, you can pass one or more parameters to the target process separated by commas. For example:

```
Calculate lMyScript as "/Users/user_1/my_script.sh"
Launch program /System/Applications/Utilities/Terminal.app,[lMyScript] (Do not quit Omnis)
```

Line:

Command group	Flag affected	Reversible	Execute on client
Text	NO	NO	YES

Syntax

Line: *line-text*

Description

Adds a line of text to the text buffer for the current method stack. The **Line:** command supports leading and trailing spaces and can contain square bracket notation, that is, you can include or add the contents of a variable to the text buffer. You build up the text block using the Begin text block and any combination of one or more *Text:* or **Line:** commands. The Carriage return and Linefeed options of the *Begin text block* command specify the line delimiter added to the text buffer after the text added by the **Line:** command. When you have placed one **Line:** command and you press Ctrl/Cmnd-N to create a new method line, a new **Line:** command is added. You should end a block of text with the End text block command, and you can return the contents of the text buffer using the Get text block command.

See the *Text:* command which can have line specific options, unlike Line:.

Load connected records

Command group	Flag affected	Reversible	Execute on
Finding data	YES	YES	NO

Syntax

Load connected records {*file-name*}

Description

This command loads the connected records for the specified file. The **Load connected records** command ensures that the identity of the current connected records for the current record is correct. As Omnis automatically loads connected records of the main file into the current record buffer, this command is not usually required. However, in multi-user systems, this command ensures that, if any other workstation makes changes to the way in which records are connected, these changes will be reflected at the current workstation.

The flag is cleared if there is no current record for the specified file class, and in the event that no file class is specified, Omnis uses the main file. This command does not clear the Prepare for update mode but does cause multi-user semaphores to be set and should be avoided when in Prepare for... mode.

If a parent record requires locking, another user is editing it, and the Wait for semaphores command is on, the lock cursor will be displayed. If the user cancels the lock, the flag is cleared and the parent record is not loaded. The Do not wait for semaphores command prevents the user from having to wait for the record and returns a flag false if the parent record is not available.

If placed in a reversible block, the parent record reverts to its former value when the method terminates. If you need to read in grandparent records, you can add this command to the usual Next command:

Example

```
# Use load connected records to load the grandparent record,

# as only the parent record of the main file is loaded after a find
Set main file {fChild}
Find first
Load connected records {fParent}

Do $cinst.$redraw()
```

Load error handler

Command group	Flag affected	Reversible	Execut
Error handlers	NO	YES	NO

Syntax

Load error handler ([All libraries]) [name/]name (first-error-number, last-error-number)

Options

All libraries	If specified, the error handler applies to errors encountered in all libraries, rather than just the calling library

Description

This command loads a specified method which handles errors which may occur within a library. You can specify a range of error codes to be handled by the handler by giving the first and last error number. If no range is specified, the handler is called for all errors. Errors are either Fatal or Warning.

Error codes such as kerrUnqindex, kerrBadnotation, kerrSQL, can also be used as parameters. The Catalog window lists all the constants available in Omnis.

Fatal errors

A fatal error is one that normally stops method execution and drops into the debugger if available. The error code #ERRCODE is displayed on the status line in the debugger and is greater than 100,000.

Warning errors

A warning error is one that does not normally quit the method nor report an error description. The error code #ERRCODE is displayed on the status line in the debugger, if invoked, and is less than 100,000.

The check box option All libraries is provided. If this is not checked, the handler is called only for errors encountered in the library which loaded the error handler. This command leaves the flag unaffected and is reversible; that is, the handler is unloaded when the command is reversed. An error handler remains loaded until it is unloaded or the library containing the handler method is closed. Error handlers loaded within an error handler always unload when that error handler terminates.

An alternative to using the parameters passed to the error handler, is to use the variables #ERRCODE and #ERRTEXT. However, you must copy the values of #ERRCODE and #ERRTEXT upon entry to the error handler, since commands you execute in the error handler might change their values.

An error handler can use one of the Set error action commands (SEA) to set what it requires the next action to be. If the error handler quits without making a Set error action and there is another handler capable of accepting the error, the second handler is called. Otherwise, the default action for the error is carried out, depending on whether it is a fatal error or warning.

If an error occurs within an error handler, that error is handled in the usual way except that the original error handler will not be used (even if it could handle that error). It is possible to load error handlers within an error handler; these are meant to deal with errors within the handler and are unloaded automatically when the error handler completes execution.

Example

```
# pCode is defined as a Long Integer

# pText is defined as a character type
# A typical error handler
If pCode=kerrBadnotation
  # handle error - pText contains a string describing the error

End If
# The following example handles the error returned by the data manager when an attempt to

# duplicate a unique index occurs on update:
Load error handler cMyErrorHandler/Errors
Prepare for edit
Enter data

Update files if flag set
# In the method Errors of code class cMyErrorHandler
If pCode=kerrUnqindex
  OK message Error (Icon) {You have entered a duplicate field value/'X' has been appended to your entry}
  Calculate iValue as con(iValue,'X')
  Enter data
  If flag true
    SEA repeat command
  Else
    SEA con execution
  End If

End If
```

Load event handler

Command group	Flag affected	Reversible	Execute on client
Externals	YES	YES	NO

Syntax

Load event handler routine-name or library-name/routine-name (parameters)

Description

This command makes the specified external routine an event handler, enabling the routine to show its own windows, put its own menus on the menu bar, act as its own event filter, and so on.

Event handlers are modules of code which, when loaded, form part of the Omnis event-processing loop. Events are passed to the external before being handled by Omnis. As each call to the external takes place, it can identify whether to take appropriate action. If the event handler returns a flag false, Omnis knows that the event was meant for Omnis and the external has ignored it.

You can enter the routine name as the parameter. If the library/resource is not in the EXTERNAL folder, the name of the file containing the library/resource and the name of the library/resource within that file are given as parameters. If no file name is given, the current dynamic link library/resource is searched for the specified routine name.

When the method is called, any existing event handler is not unloaded but continues to be called along with the new handler. The flag is cleared if the routine cannot be loaded.

If you use **Load event handler** in a reversible block, the event handler is unloaded when the method containing the reversible block terminates.

You can pass parameters to the external code by enclosing a comma-separated list of fields and calculations. If you pass a field name, for example, Call external routine Maths1 (LVAR1,LVAR2), the external can directly alter the field value. Enclosing the field in brackets, for example, Call external routine Maths1 ((LVAR1),(LVAR2)), converts the field to a value and protects the field from alteration.

In the routine itself, the parameters are read using the usual GetFldVal or GetFldNval with the predefined references Ref_parm1, Ref_parm2, and so on, Ref_parmcnt gives the number of parameters passed. If the field name is passed as a parameter, you can use SetFldVal or SetFldNval with Ref_parm1, and so on, to change the field's value.

Example

```
Load event handler myEventHandler
```

Load external routine

Command group	Flag affected	Reversible	Execute on client
Externals	YES	YES	NO

Syntax

Load external routine routine-name or library-name/routine-name (parameters)

Description

This command loads the specified external code into memory. You can enter the routine name as the parameter. If the library/resource is not in the EXTERNAL folder, the name of the file containing the library/resource and the library/resource name within that file are given as parameters.

If the library/resource is already loaded or is not found, the flag is cleared and no action is taken. If this command is included in a reversible block, the library/resource is unloaded when the method terminates. If the library/resource is loaded in, it is called with the mode set at ext_load.

You can pass parameters to the external code by enclosing a comma-separated list of fields and calculations. If you pass a field name, for example, Call external routine Maths1 (LVAR1,LVAR2), the external can directly alter the field value. Enclosing the field in brackets, for example, Call external routine Maths1 ((LVAR1),(LVAR2)), converts the field to a value and protects the field from alteration.

In the routine itself, the parameters are read using the usual GetFldVal or GetFldNval with the predefined references Ref_parm1, Ref_parm2, and so on, Ref_parmcnt gives the number of parameters passed. If the field name is passed as a parameter, you can use SetFldVal or SetFldNval with Ref_parm1, and so on, to change the field's value.

Example

```
Load external routine MathsLib/sqr (iNumber,iNumber2)
```

Load from list

Command group	Flag affected	Reversible	Execute on client
List lines	YES	NO	NO

Syntax

Load from list {*line-number* (*variable-names*) {default is current line}}

Deprecated Command

This command has been deprecated and is no longer visible in the Code Assistant in the Code Editor (it will not appear when you type the first few characters), although it is still present in Omnis Studio and will continue to function if used in legacy code. You can show this command by disabling the appropriate Command Filter in the **Modify** menu in the Code Editor.

Description

This command transfers field values from the current list to the corresponding fields in the current record buffer. However, if you include a list of fields, the values in the current list are transferred to the specified fields (see example). Each column value, taken in the order it was defined, is copied to the corresponding field in the field list.

Field names parameter list

The command **Load from list** with '(CVAR1„CVAR12)' specified will load the first column of the current line of the list into CVAR1, ignore the second column, and load the third column into LVAR12. If too few field names are specified, the other columns are not loaded. If too many field names are specified, the extra fields are cleared. Any conversions required between data types are carried out.

If the line number specified in the command line is empty, or if it evaluates to zero, the values are loaded from the current line. If the list is empty or if the line evaluates to a value greater than the total number of lines in the list, the flag is cleared and the fields in the parameter list or in the list definition are cleared.

Example

```
Set current list lMyList
Define list {lName,lAge}
Add line to list {('Fred',10)}
Add line to list {('George',20)}
Add line to list {('Harry',22)}
Add line to list {('William',31)}

Add line to list {('David',62)}
Do lMyList.$line.$assign(4) ## set the current line
Load from list ## load the values from the current line into lName and lAge
Load from list {2} ## load the values from line 2 into lName and lAge
Load from list {4 (lTmpName,lTmpAge)}

# load the values from line 2 into lTmpName and lTmpAge
```

Load page setup

Command group	Flag affected	Reversible	Execute on client
Reports and Printing	NO	YES	NO

Syntax

Load page setup

Description

This command loads the page setup from the current report class and makes it the current page setup. Every report class has optionally a page setup stored with it, for use when the report is printed. The flag is set if there is a current report class and it contains a page setup. When used in a reversible block the previous page setup is restored once the method has finished.

The stored page setup for a report class never becomes the current page setup unless a **Load page setup** command is issued.

Example

```
# Load the page setup for rMyReport
Set report name rMyReport
Load page setup

Print report
```

MailSplit

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

MailSplit (*message*,*headerlist*,*body*{Char|Bin|MIME-List}) **Returns** *status*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

MAILSplit parses an Internet e-mail message. It can also decompose MIME content. It returns a Status value less than zero if an error occurs. Possible error codes are listed in the Web Command Error Codes Appendix.

Message is an Omnis Binary or Character 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, and by the commands POP3RecvMessage and IMAPRecvMessage. You can also pass the headers returned by POP3RecvHeaders and IMAPRecvHeaders, in order to parse the headers.For correct results with many of the encodings supported by **MAILSplit** you must use a Binary field to receive the message.

Example message:

```
Received: by omnis.net 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
```

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.net with SMTP; 12 Aug 1996 11:49:59 –0700

Attribute	Value
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

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: Omnis Software. 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.

Alternatively, 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.

Header Values Containing International Characters

MAILSplit supports RFC 2047, for the UTF-8 and ISO8859-N character encodings. When it encounters text in header values that is encoded according to the RFC 2047 rules for the character encodings it supports, it converts the header value into its original value before storing it in the HeaderList.

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 up to nine columns (the last three columns are optional):

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.

Column	Contains
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 Macintosh, character data in the iso-8859-1 character set has been converted to the Macintosh character set. On all platforms, the command replaces CRLFs with the Omnis newline character.
Content-disposition	The content disposition of the body part. Either empty, “attachment” or “inline”. This is a hint to the receiving application about how to handle the content. Inline body parts are intended to be displayed when the message is displayed, whereas attachments are considered separate from the main body of the mail message, and their display should not be automatic.

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
0	multipart	Alternative			
1	text	Plain		From Bob	
1	text	Html		<!DOCTYPE HTML...	

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
0	multipart	mixed			
1	multipart	alternative			
2	text	plain		From Bob	
2	text	html		<!DOCTYPE HTML...	
1	application	octet-stream	App.h		This is my

Example

```
# Split and decompose pMessage as received from POP3Recv,POP3RecvHeaders,POP3RecvMessage,
# IMAPRecvHeaders or IMAPRecvMessage
# Return pDate, pFrom, pSubject and pBody (if message rather than headers) and save
# any attachments in pEnclosurePath
Do lHeaderList.$define(lAttribute,lValue)
Do lMimeList.$define(lLevel,lContentType,lContentSubType,lFileName,lCharData,lBinData,lCharSet,lEncoding)
MailSplit (pMessage,lHeaderList,lMimeList)
# extract header information
Do lHeaderList.$search(upp(lAttribute)='DATE'|upp(lAttribute)='FROM'|upp(lAttribute)='SUBJECT')
Do lHeaderList.$first(kTrue,kFalse) Returns lLineRef
While lLineRef
  Do lHeaderList.$loadcols()
  Switch upp(lAttribute)
    Case 'DATE'
      Calculate pDate as lValue
    Case 'FROM'
      Calculate pFrom as lValue
    Case 'SUBJECT'
      Calculate pSubject as lValue
  End Switch
  Do lHeaderList.$next(lLineRef,kTrue,kFalse)
End While
# decompose the MIME content from lMimeList
```

```
For lMimeList.$line from 1 to lMimeList.$linecount step 1
  Do lMimeList.$loadcols()
  If lContentType='text'&(lContentSubType='plain') ## found body of e-mail in character format.
    Calculate pBody as lCharData
  End If
  If lFileName<>'' ## found file attachment, write the file to the enclosures folder
    Calculate lFilePath as con(pEnclosurePath,lFileName)
    Do lFileOps.$createfile(lFilePath)
    Do lFileOps.$openfile(lFilePath)
    Do lFileOps.$writefile(lBinData) Returns lReturnFlag
    Do lFileOps.$closefile()
  End If
End For
```

Maximize window instance

Command group	Flag affected	Reversible	Execute on client
Windows	NO	NO	NO

Syntax

Maximize window instance *window-instance-name*

Description

Maximizes the specified window instance.

Example

```
# Maximize the window wMyWindow to full screen
Maximize window instance wMyWindow
# Alternatively, you can do it like this
Do $cwind.$maximize()
# Or like this
Do $iwindows.wTest.$bringtofront(kTrue) ## if kTrue, the window instance is brought to the front restoring pos
```

Merge list

Command group	Flag affected	Reversible	Execute on client
Lists	YES	NO	NO

Syntax

Merge list list-or-row-name ([Clear list][,Use search])

Options

Clear list	If specified, the command empties the current list and defines it to match the specified list before executing
Use search	If specified, the command uses the current search to select data

Deprecated Command

This command has been deprecated and is no longer visible in the Code Assistant in the Code Editor (it will not appear when you type the first few characters), although it is still present in Omnis Studio and will continue to function if used in legacy code. You can show this command by disabling the appropriate Command Filter in the **Modify** menu in the Code Editor.

Description

This command adds the specified list to the end of the list previously specified as the current list. Once the list reaches its maximum size, the command finishes and clears the flag. Omnis does not check that the same fields are stored in the two lists (which they should be). If the same fields are not present, data is not transferred.

If you use the *Clear list* option, the current list is initially cleared and defined to hold the same fields as the specified list. This is the same as copying a list.

If you use the *Use search* option, only lines matching the search class are merged or added to the current list. All lines match if there is no current search class.

Example

```
# To merge the list iList1 to the current list iList2
Set current list iList2
Set search name sMySearch
Merge list iList1 (Clear list ,Use search)
If flag true
  Sort list
Else
  OK message {Merge failed at line [iList1.$linecount]}
End If
# To append only selected lines
Set current list iList2
Set search as calculation {#LSEL}
Merge list iList1 (Use search)
# or do it like this
Do iList2.$merge(iList1)
```

Message timeout

Command group	Flag affected	Reversible	Ex
Exchanging data	NO	NO	NO

Syntax

Message timeout {*interval* (seconds)}

Description

This command specifies the time Omnis has to wait for DDE responses to messages sent to other applications. There is a default value of 30 seconds when Omnis is started.

The following general purpose method sets up a DDE channel by increasing the message timeout by 5 seconds until successful. You pass three parameters to the method, that is, the initial timeout, the channel number and the program 'name|document'.

Example

```
# open dde channel
# parameter pNum is short int
# parameter pChannel is short int
# parameter pProgDoc is character
Set DDE channel number {pChannel}
Repeat
  Message timeout {pNum}
  Open DDE channel {[pProgDoc]}
  If flag false
```

```
    Yes/No message {Give up 'Open DDE channel'?}
    If flag true
      Close DDE channel
    End If
  End If
  Calculate pNum as pNum+5
Until flag true
```

Minimize window instance

Command group	Flag affected	Reversible	Execute on client
Windows	NO	NO	NO

Syntax

Minimize window instance *window-instance-name*

Description

This command minimizes the specified window instance:

- On Windows and Linux, the window is shown as an icon at the bottom of the Omnis application window.
- On macOS, the window is shown as an icon in the dock.

Example

```
# Minimize the window wMyWindow to reduce it to an icon
Minimize window instance wMyWindow
# Alternatively, you can do it like this
Do $cwind.$minimize()
```

Modify class

Command group	Flag affected	Reversible	Execute on client
Classes	YES	NO	NO

Syntax

Modify class {*class-name*}

Description

This command opens a library class in design mode. Method execution continues and does not wait for the design window to be closed. **Modify class** lets users modify new search and report classes created with the New class command. Opening a class in design mode when one of its methods is running causes a Quit all methods to be carried out before the design window opens. If the class does not exist, the command clears the flag.

Example

```
New class {Search Class/sOverDrawn}
Modify class {sOverDrawn}
# now you can
Set search name sOverDrawn
Print report (Use search)
```

Modify methods

Command group	Flag affected	Reversible	Execute on client
Classes	YES	NO	NO

Syntax

Modify methods {*class-name*}

Description

This command opens the method editor for the specified class. Method execution continues and does not wait for the design window to be closed. Opening a method in design mode first causes a Quit all methods if one of the methods for that class is running. The flag is cleared if the specified class does not exist, or if it is a file, search, or report class.

Example

```
New class {Window/wMyWindow}
# open at the $construct() method for the window wMyWindow
Modify methods {wMyWindow}
```

Move file

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

Move file (*from-path*, *to-path*) **Returns** *err-code*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This command moves the file specified in from-path to the folder named in to-path. It returns an error code (See Error Codes), or zero if no error occurs. The to-path is the path to destination folder into which the file will be moved. The command may fail if the to-path directory contains a file with the same name as from-path filename.

Move file cannot move a file across volumes (disks). Use Copy file and Delete file instead. **Move file** cannot move directories.

Example

```
# Prompt the user for a file to move together with a path
# to move to and move the file
Do FileOps.$putfilename(lPathname,'Select a file for moving','') Returns lReturnFlag
If lReturnFlag
  Do FileOps.$selectdirectory(lNewPath,'Path to move to') Returns lReturnFlag
  If lReturnFlag
    Move file (lPathname,lNewPath) Returns lErrCode
  End If
End If
```

New class

Command group	Flag affected	Reversible	Execute on client
Classes	YES	NO	NO

Syntax

New class {superclass-name or class-type/name}

Description

This command creates a new class with the specified type and class name. For example, you can use New class in association with Modify class to allow users to create new search and report classes. Attempting to create a class with the same name as one which already exists clears the flag and displays an error message.

Example

```
New class {Window/wMywindow}
Modify class {wMyWindow}
```

Next

Command group	Flag affected	Reversible	Execute on
Finding data	YES	YES	NO

Syntax

Next on field-name ([Exact match][,Use search])

Options

Exact match	If specified, the index value of the field in suitable records must equal the current value
Use search	If specified, the command uses the current search to select data

Description

This command locates the next record using the current find table. The **Next** command works in the same way as the corresponding option on the Commands menu but with no redraw, allowing you to work through a file. It is usually used after a Find command which creates a find table of records.

If the Index field, *Exact match* and/or Search option used in the **Next** is incompatible with the preceding Find, a new find table is built. Normally, the parameters in this command are left blank so that the current find table is used.

If the **Next** command does not follow a Find, a find table is built for the current main file before doing the **Next**.

If an indexed field is specified, **Next** on SU_NAME for example, the find table is just the index order for the field. The *Use search* option creates a find table for the current main file in which the search specification is implicitly stored. Thus, changes to the search do not affect the find table once it is created.

Once the next record is located, the main and connected files are read into the current record buffer.

An error occurs whenever a **Next** on FIELD command is performed on a non-indexed field or if the field is not in the main file or its connected files.

If the next record is found, the flag is set; if not, it is cleared.

If the *Exact match* option is chosen, the next record is loaded where the index value of the specified field matches the current value.

If you use **Next** with a search, it builds a find table if necessary and finds the next record listed on the find table which meets the search criteria.

Example

```
# Add 5% to all account balances
Find first on fAccounts.Code
While flag true
  Calculate fAccounts.Balance as fAccounts.Balance+((fAccounts.Balance/100)*5)
  Update files
  Next
End While
```

No/Yes message

Command group	Flag affected	Reversible	Execute c
Message boxes	YES	NO	NO

Syntax

No/Yes message title ([Icon][,Sound bell][,Cancel button]) {message}

Options

Icon	If specified, the message displays an operating system specific icon
Sound bell	If specified, the system bell sounds when the command displays the message
Cancel button	If specified, the message has a cancel button

Description

This command displays a message box containing the specified message and provides a No and a Yes pushbutton. You can include a Cancel button by checking the *Cancel button* option. When the message box is displayed method execution is halted temporarily; it remains open until the user clicks on one of the buttons before continuing. The No button is the default button and can therefore be selected by pressing the Return key.

The number of lines displayed in the message box depends on your operating system, fonts and screen size. In the message text you can force a break between lines (a carriage return) by using the notation "//"or the kCr constant enclosed in square brackets, e.g. 'First line[kCr]Second line'. Also you can add a short title for the message box.

For greater emphasis, you can select an Icon for the message box (the default "info" icon for the current operating system), and you can force the system bell to sound by checking the *Sound bell* check box. Under Windows XP, you have to specify a system sound for a 'Question' in the Control Panel for the *Sound Bell* option to work.

You can insert a **No/Yes message** at any appropriate point in a method. If the user clicks the No button, the flag is cleared; otherwise, a Yes sets the flag. You can use the msgcancelled() function to detect if the user pressed the Cancel button.

Example

```
# Open a No/Yes dialog and display the option selected
No/Yes message My Editor (Icon,Cancel button) {Do you wish to save the changes you have made ?}
If msgcancelled()
  OK message My Editor {Cancel button pressed}
Else
  If flag true
    OK message My Editor {OK button pressed}
  Else
    OK message My Editor {Cancel button pressed}
  End If
End If
```

OK message

Command group	Flag affected	Reversible	Execute c
Message boxes	NO	NO	YES

Syntax

OK message title ([Icon][,Sound bell][,Cancel button]) {message}

Options

Icon	If specified, the message displays an operating system specific icon
Sound bell	If specified, the system bell sounds when the command displays the message
Cancel button	If specified, the message has a cancel button

Description

This command displays the specified message and waits for the user to click the OK or Cancel button before continuing. Method execution is halted temporarily while the message box is displayed. Note: for JavaScript client-executed methods this command uses a standard alert() or confirm() dialog.

The number of message lines displayed depends on your operating system, fonts and screen size. In the message text you can force a break between lines (a carriage return) by using the notation "//" or the kCr constant enclosed in square brackets, e.g. 'First line[kCr]Second line'. Also you can add a short title for the message box.

For greater emphasis, you can pass the Icon option to add an icon (the default "info" icon for the current operating system). If no icon is specified, the default action on macOS is to show the application icon (applies to Big Sur or later). You can force the system bell to sound by passing the Sound bell option.

The message box displayed by this command has an OK button by default, but you can add a Cancel button by passing the Cancel button option. After executing an **OK message**, the flag is unchanged, but you can use the msgcancelled() function to detect if the user pressed the Cancel button.

You can use square bracket notation in the message text to display the current value of fields and variables.

Example

```
'omnis # Open a Ok messsage dialog, if cancel is pressed abort printing Calculate lUserName as 'My Name'
OK message My Editor (Icon,Cancel button) {Ready to print, press Ok to continue} If msgcancelled()   OK
message My Editor {Printing aborted by user [lUserName]}   Quit method End If1'
```

On

Command group	Flag affected	Reversible	Execute on client
Events	NO	NO	YES

Syntax

On *event-code* or *codes* (code1,code2,...)

Description

This command is used in an event handling method and marks the beginning of a code segment that executes when the specified event (or one of a number events) is received by the current event handling method. An **On** command also marks the end of any preceding **On** statement. You specify the event or list of events using the event constants.

When Omnis generates an event it sends the event information as a series of event parameters to the appropriate event handling method. The first parameter is always an event constant. Further parameters, if any, depend on the event and further describe the event. This event information is interpreted by the **On** statements in the event handling methods. Window field events are sent to the $event() method behind the field, then to the $control() method for the window instance, and then to the $control() method for the current task. Events that occur in the window itself, such as a click on the window background, are sent to the class method called $event(), then to the $control() method for the current task. A particular event is sent to the first **On** command which applies, and when the next **On** command is encountered quits the method.

You should place any code which is to be executed for all events before the first On command. You cannot nest **On** commands or put them in an If or Else statement. You can use On default to handle any events not handled by an earlier **On** event command. The **On** commands must be in event handling methods only: if used elsewhere they are not executed. The function sys(86) at the start of a method reports any events received by the object.

See also Quit event handler.

Example

```
# This example shows typical event handling for a field
On evBefore
# code to process an evBefore event

On evAfter
# code to process an evAfter event

On evClick,evDoubleClick
# code to process both evClick and evDoubleClick events
```

On default

Command group	Flag affected	Reversible	Execute on client
Events	NO	NO	YES

Syntax

On default

Description

This command is used in an event handling method and handles any events not handled by the preceding On commands. You use the On command to mark the beginning and end of an On statement. You should place any code which is to be executed for all events before the first On command.

Example

“‘omnis On evClick # process code for evClick event

On default # handle all other events “‘

Open check data log

Command group	Flag affected	Reversible	Ex
Data management	NO	YES	NO

Syntax

Open check data log ([*Do not wait for user*])

Options

Do not wait for user	Unless this option is specified, the user must close the window before method execution continues, and before doing anything else

Description

This command opens the check data log. If the Do not wait for user option is specified, execution continues with the next command, otherwise execution stops until the user has closed the log. You use the check data log to manage the problems encountered in a data file after the Check data command is run. The data log window lets you repair any problems listed in the window, print the contents of the log, or clear the log.

Example

```
Check data (Check indexes)
Open check data log
```

Open data file

Command group	Flag affected	Reversible	Execute on client
Data files	YES	NO	NO

Syntax

Open data file ([Do not close other data][,Read-only][,No conversion by runtime][,Convert without user prompts][,Full Unicode conversion]) {file-name[,internal-name] or odb://[address:port:]name[,internal-name]}

Options

Do not close other data	If specified, the command does not close all open data files before opening the specified data file
Read-only	If specified, the data file is opened in read-only mode
No conversion by runtime	Omnis normally offers to convert data files created by an earlier version of Omnis. If this option is specified, the runtime version of Omnis will not offer to convert the file, and the command will fail
Convert without user prompts	If specified, and conversion is allowed, Omnis will immediately perform the conversion without giving the user any prompts that require a response; also, the user cannot cancel the conversion
Full Unicode conversion	Unicode Studio only. If specified, and convert without user prompts is specified, do full Unicode conversion instead of quick conversion (quick conversion is only ok when you know all character data in the file is 7 bit)

Description

This command opens the specified Omnis data file (.df1 file) and makes that file the "current" data file, using either the pathname of the datafile, or the location of the datafile hosted via the Omnis Data Bridge (ODB). It clears the flag if the data file cannot be found or opened. If the *Do not close other data* check box option is not specified, all existing data files are closed even if the command fails. Opening a data file which is already open will close and reopen that data file. The *Read-only* Studio/Omnis 7 check box causes the data file to be opened in read-only mode. This lets you open an Omnis 7 data file in read-only mode in Omnis Studio without conversion taking place.

If you select the *No conversion by runtime* option, and the data file was created with a previous version of Omnis, then the runtime version of Omnis will not convert the data file. The default is that an Omnis runtime will ask the user if they want to convert the data file.

If an opened data file uses more than one segment, all segments are opened. The rules for finding the additional segments which form part of the data file are as follows:

- Under Windows and Linux, the paths given in the Omnis environment variable are searched; if the file is not in any of these locations, then Omnis searches the directory containing the first segment.
- Under macOS, root directories of all mounted volumes are searched as well as the folders containing the first segment and the most recently opened library.

You can override the default internal name by specifying your own in the parameter for the command.

If the data file is to be accessed using the Omnis Data Bridge (ODB), then instead of using a pathname, you can specify the location of the file using a special syntax:

- odb://[address:port:]name

where *address:port* is the TCP/IP address and port number of the ODB server, e.g. 127.0.0.1:5900, and *name* is the name of a data file accessed using the ODB server. You can omit address:port:, in which case Omnis uses the address and port stored in the $odbserver root preference. Note that the value of $odbserver is stored in the file odb.txt in the studio folder of the Omnis installation tree.

Example

```
Open data file {Sales.df1,Sales}
If flag true
  Find first
  If flag true
    Open data file (Do not close other dat) {Purch.df1,Purchases}
    If flag true
      Calculate fPurchases.Field1 as fSales.Field1
      Prepare for insert with current values
      Enter data
      Update files if flag set
    End If
  End If
End If

# Example 2 - Transfer datafile 1 to datafile 2
Open data file {pOrders.df1,pOrders1}
If flag true
  Set main file {fOrders}
  Find first on fOrders.OrderNum
  While flag true
    Prepare for insert with current values
    Open data file {pOrders2.df1,pOrders2}
    Update files if flag set
    Open data file {pOrders.df1,pOrders1}
    Next on fOrders.OrderNum
  End While
End If
```

```
# Example 3 - Open a data file on a specific ODB server
Open data file {odb://127.0.0.1:5900:test}

# Example 4 - Open a data file using the ODB server identified by $prefs.$odbserver
Open data file {odb://test}
```

Please note:

- When the Open data file command is used with an odb:// prefix and an *internal-name* parameter, a comma separator is still required, for example:

```
Open data file odb://127.0.0.1:5900:test, myDatafile
```

- In Studio 10.0 and above, curly braces are not required around the command parameters.

Open DDE channel

Command group	Flag affected	Reversible	Ex
Exchanging data	YES	YES	NO

Syntax

Open DDE channel {*program-name* | *topic-name*}

Description

This command opens the current channel for exchanging data. If there is a valid response, the flag is set and the channel is successfully opened. If the channel is already open, the existing conversation is closed.

When entering the command in a method, you use the parameters to specify the program and the topic to which the message is to be addressed. Note that the "pipe" (or vertical bar) between the program name and topic name is required.

This command is reversible, that is, a previous conversation will reopen if this command is contained within a reversible block.

When the command is used in a method containing a reversible block, and if a new conversation is initiated using the same channel number as an existing conversation, the original continues to process incoming messages only, and at the end of the method, the new conversation is stopped and the original becomes fully active.

Example

```
Set DDE channel number {2}
Open DDE channel {Omnis|Country}
If flag false
  OK message {The Country library is not running}
Else
  Do method TransferData
  Close DDE channel
  OK message {Update finished}
End If
```

Open file

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

Open file (*path*, *refnum* [,*'r'*]) **Returns** *err-code*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

It returns an error code (See Error Codes), or zero if no error occurs.

Example

```
# Prompt the user for a file for opening
Do FileOps.$putfilename(lPathname,'Select a file','') Returns lReturnFlag
If lReturnFlag
  Open file (lPathname,lRefNum)
End If
```

Open library

Command group	Flag affected	Reversible	Execute on client
Libraries	YES	NO	NO

Syntax

Open library ([Do not close others][,Enable conversion by runtime][,Do not open startup task][,Convert without user prompts]) library-file-name,internal-name,password (parameters)

Options

Do not close others	If specified, the command does not close all open libraries before opening the specified library
Enable conversion by runtime	The development version of Omnis offers to convert libraries created by an earlier version of Omnis. If this option is specified, the runtime version of Omnis will also offer to convert such libraries
Do not open startup task	If specified, the command does not construct an instance of the startup task when it opens the library
Convert without user prompts	If specified, and conversion is allowed, Omnis will immediately perform the conversion without giving the user any prompts that require a response; also, the user cannot cancel the conversion

Description

This command opens the specified library file and closes other libraries, if specified. You specify the library name (including path name if required), internal name, password, and startup method parameters of the library to be opened. If the disk file with the specified path name cannot be opened or is not a valid library, the flag is cleared and no libraries are closed.

If the *internal name* of an opened library is specified, a check is made to ensure the internal name is unique among the open libraries, and a runtime error occurs if this is not the case. If no internal name is specified, the default internal name is the disk name of the file with the path name and suffix removed. For example, the internal name for 'hd:myfiles:testlib.lbs' is 'testlib'.

Do not close others

The *Do not close others* option lets you keep open all other libraries. Otherwise, all other open libraries are closed (see the Close library command for the consequences of closing a library). If an attempt is made to open a library which is already open, that library is closed and reopened.

Startup task

If the *Do not open startup task* option is specified, the startup task construct for the opened library is not called. Otherwise, the startup task $construct() method is called and the parameters for it are passed. The startup task instance name will be either the library name or the library internal name if it has one: it is not called Startup_Task.

Enable conversion by runtime

If you select the *Enable conversion by runtime* option, and the library was created with a previous version of Omnis, then the runtime version of Omnis can convert the library if the user allows. The default is that an Omnis runtime will not ask the user if they want to convert the library.

Passwords

If a *password* is specified, an attempt is made to open the library with that password. If it is not a valid password or no password is specified, the library is opened in the usual way, that is, if the library does not need a master password, it is opened at the master level; otherwise, the usual prompt for password dialog is opened (the library is closed and a flag false returned if this dialog is closed without a password being entered).

Example

```
# Open the library mylib.lbs from the root of your
# omnis studio tree
Calculate lLibPath as con(sys(115),'mylib.lbs')
Open library (Do not close others) [lLibPath],MYLIB
If flag true
  OK message {Library Opened!}
End If
```

Open lookup file

Command group	Flag affected	Reversible	Execute on clien
Data files	YES	YES	NO

Syntax

Open lookup file {lookup-name,data-file-name,file-class-name,index-field} **Open lookup file** lookup-name,data-file-name,file-class-name,index-field (Studio 10 and later)

Description

This command opens an Omnis data file for use as a lookup file. You give each lookup file a reference name which you use in subsequent lookup() functions to select a particular data file and file class. You can open any Omnis data file as a lookup file, including any data file accessed via the Omnis Data Bridge (ODB).

In a lookup file, you can use the file classes to look up field values based on an indexed search. Each file class should consist of at least two fields: the first is the index (usually a character field), the second is any field type. For example, the data file Lookup.df1 has a file class called fCities with the following structure:

File name	Field1	Field2
fPic	Char Indexed	Picture
fCities	Char Indexed	Char

The parameters for **Open lookup file** are separated by ",". The first parameter is a label that you create to become the reference to that lookup "channel". If you omit this label, Omnis assumes that you will use only one lookup file whereupon you can use lookup() without its first parameter. The label you give to each lookup is case-insensitive and if you use the same one twice, the previous lookup file is closed. A flag true is returned if the data file is found and opened.

The example at the bottom opens a data file called Lookup.df1 and assigns the label "City" to the lookup channel. The City lookup uses the file class fCities within that data file and uses the first index to search for the required data. The OK message uses lookup() to search the first indexed field for an exact match with the value "I". If the match is found, the value of field 2 in the matched record is returned and displayed as part of the OK message. If no match is found, lookup() returns an empty value.

Note that the index and field are specified as *numbers* because your particular library may not include the file class used in the lookup data file. If you omit either number, the default is to use the first field as the index, and the second as the field value to be returned in the lookup() function.

Omnis opens the data file using the following rules. Omnis first tries to open the file using the supplied *data-file-name*. If this fails, and if the *data-file-name* does not contain any special characters used in pathnames (for example, under Windows ':' and '\'), then Omnis searches for the file.

Under Windows and Linux, Omnis searches the paths included in the Omnis environment variable. The Omnis environment variable must contain a semicolon separated list of pathnames, for example:

```
C:\OMNIS\LOOKUPS#D:\OMNIS\LOOKUPS
```

Under macOS, Omnis searches the System folder, Omnis folder and then the root of each mounted volume, in that order.

The flag is set if the lookup is successful, that is, the data file is opened, the file slot exists and the indexed field is indeed indexed. The lookup file is closed if the command is reversed (see **Begin reversible block**).

You can close lookup files using **Close lookup file**, but this is not necessary: all lookup files associated with a library are closed automatically when the library quits.

You can maintain the data within the lookup file from within the library by:

- Adding the appropriate file classes to your library,
- Changing the data file to the lookup file using Open data file,
- Opening a window and editing/ inserting data in the usual way, and
- Returning to the original data file.

You can also load multiple data files with Open data file.

Example

```
Open lookup file {City,Lookup.df1,fCities,1}
If flag true
  OK message {The city you require is [lookup('City','I',2)]}
End If
# You can open more than one file class within a particular data file by assigning a different label to each l
Open lookup file {City2,Lookup.df1,fCities2}
Open lookup file {City,Lookup.df1,fCities}
Open lookup file {Country,Lookup.df1,fCountries}
# You can also open a lookup file accessed using the Omnis Data Bridge (ODB)
Open lookup file {City,odb://127.0.0.1:5900:LookUpData,fCities,1}
```

Open runtime data file browser

Command group	Flag affected	Reversible	Ex
Data management	NO	NO	NO

Syntax

Open runtime data file browser

Description

Example

```
Open data file {Salaries.df1}
Set current data file {Salaries}
Open runtime data file browser
```

Open task instance

Command group	Flag affected	Reversible	Execute on client
Tasks	NO	NO	NO

Syntax

Open task instance class-name[/instance-name] [(parameters)]

Description

This command opens the specified task and assigns an instance name. You can include a list of parameters which are sent to the $construct() method in the task instance. Note that startup task instance is normally opened when the library opens: its name will be either the library name or the library internal name if it has one.

Example

```
Open task instance tkMyTask (1)
# or do it like this
Do $tasks.tkMyTask.$open('*',1) ## * is the default instance name
# Then in the $construct of tkMyTask
If pOpenWindow ## pOpenWindow is a boolean parameter variable
  Open window instance wMyWindow
End If
```

Open trace log

Command group	Flag affected	Reversible	Execute on client
Debugger	NO	NO	NO

Syntax

Open trace log ([*Clear trace log*])

Options

Clear trace log	If specified, the command clears the trace log

Description

This command opens the trace log. The trace log can also be opened via the Tools menu.

Example

```
# open the trace log and clear any existing messages
Open trace log (Clear trace log )
```

Open window instance

Command group	Flag affected	Reversible	Execute on client	P
Windows	NO	YES	NO	A

Syntax

Open window instance *class*[*/instance*] [*/l/t/r/b/cen/max/min/stk*] [(*params*)]

Description

This command opens an instance of the specified window class. You can specify the position and size of the window instance (using the left, top, right, bottom coordinates in pixels), and you can center, maximize, minimize, and stack the window. Furthermore, you can send a list of parameters to the window's $construct() method.

Open window instance lets you open multiple instances of the same window class. The default instance name for a window is the class name, but if you want to open multiple instances of the same window class you must assign a unique name to each instance. Window instance names are case-sensitive.

Window Position and Size

You can specify the position of the top-left corner of the window instance by adding the coordinates to the end of the window-name/instance-name parameter, that is, window-name/instance-name/left/top. You specify the position in pixels, the origin being /0/0, that is, under the menu bar. By providing all four coordinates, you can specify the position and size of the window instance.

Centering and Stacking Windows

The /CEN option automatically centers the window instance. You can include the four window size coordinates with the /CEN option so the window is sized and centered.

The /STK option opens the window instance about 12 pixels (the stack offset) to the right and down from the current top window. When a stacked window reaches the edge of the screen, it is placed back at the top of the stack, offset slightly from the first window.

Maximizing and Minimizing Windows

The /MAX option opens and maximizes the window instance. If you include the position and size coordinates with this option, the window is opened with the specified position and size and then maximized.

The /MIN option opens and minimizes the window instance. If you include the position and size coordinates with this option, the window is opened with the specified position and size and then minimized.

$construct() Method and Passing Parameters

When you open a window instance, the $construct() method for that instance is run. In this method, you place commands which set up the conditions required by the window. For example, you may want to set the main file, build particular lists, and so on. Just as with Do method and Do code method you can send parameters to the window using **Open window instance.**

Reversible blocks in the $construct() method do not reverse until the window instance is closed, unlike a normal method whose reversible blocks reverse on termination of the method.

Example

```
# Open 2 instances of the window wMyWindow stacked
Open window instance wMyWindow/wInst1/CEN
Open window instance wMyWindow/wInst2/STK

# Alternatively, you can let Omnis assign enumerated names to
# multiple instances by specifying '*' as the instance name.
Open window instance wMywindow/*
Open window instance wMywindow/*
# Specify the size and location when opening the window wMyWindow
Open window instance wMyWindow/*/10/10/100/100
# Specify the size and location in variables
```

```
Calculate lLeft as 10
Calculate lRight as 100
Calculate lTop as 10
Calculate lBottom as 100
Open window instance wMyWindow/*/[lLeft]/[lTop]/[lRight]/[lBottom]
# Open the window wMyWindow maximized
Open window instance wMyWindow/*/MAX
# Open the window wMyWindow minimized
Open window instance wMyWindow/*/MIN
# Open the window wMyWindow and pass the variables lMyVar1 and lMyVar2
# to its $construct method
Open window instance wMyWindow/*/ (lMyVar1,lMyVar2)
```

Optimize method

Command group	Flag affected	Reversible	Execute on client
Methods	NO	YES	NO

Syntax

Optimize method

Description

This command stores an optimized form of the method so when the method is executed for a second time it runs much faster. You should position this command so that it is the first executable statement of the method, except when you put it in a reversible block. Methods which are executed frequently, such as control methods and loops, are best optimized. The command is reversible and does not change the flag.

Optimize method works immediately, therefore when it is executed for the first time it converts all of the subsequent lines of the method being executed into its optimized form and continues execution. When the method terminates, the optimized form of that method is kept in RAM; the optimized form is executed if the method is called again. If **Optimize method** is in a reversible block the optimized form of the method is disposed of when the method terminates; so it will be rebuilt each time the method executes. The optimized method is also discarded whenever the design window is open for the method or the method is modified using the notation.

WARNING Optimizing too many methods will increase the memory used which may eventually result in a slowdown or worse.

Example

```
# Build a list of invoices for the first overdrawn account
Optimize method
Set main file {fAccounts}
Set current list iInvoices
Define list {fInvoices}
Set search name sOverDrawn
Find first on fAccounts.Code (Use search)
While flag true
  Single file find on fInvoices.AccCode (Exact match) {fAccounts.Code}
  Add line to list
  Next
End While
```

OR selected and saved

Command group	Flag affected	Reversible	Execute on client
List lines	YES	NO	NO

Syntax

OR selected and saved ([*All lines*]) {*line-number* (*calculation*)}

Options

All lines	If specified, the command affects all the lines in the list

Deprecated Command

This command has been deprecated and is no longer visible in the Code Assistant in the Code Editor (it will not appear when you type the first few characters), although it is still present in Omnis Studio and will continue to function if used in legacy code. You can show this command by disabling the appropriate Command Filter in the **Modify** menu in the Code Editor.

Description

This command performs a logical OR of the Saved selection with the Current selection. To allow sophisticated manipulation of data via lists, a list can store two selection states for each line; the "Current" and the "Saved" selection. The Current and Saved selections have nothing to do with saving data on the disk; they are no more than labels for two sets of selections. The lists may be held in memory and never saved to disk: they will still have a Current and Saved selection state for each line but they will be lost if not saved. When a list is stored in the data file, both sets of selections are stored.

You can specify a particular line in the list by entering either a number or a calculation.

The **OR selected and saved** command performs a logical OR on the Saved and Current states and puts the result into the Current selection. Hence, if either or both the Current and Saved states are selected, the Current state becomes selected, but if both states are deselected, the resulting Current state will remain deselected.

Logic Table (S=selected, D=deselected)

Saved	Current	Resulting Current State
S	S	S
D	S	S
S	D	S
D	D	D

The *All lines* option performs the OR on all lines of the current list. The following example selects all lines of the list.

Example

```
# Lines 3 and 5 remain selected as line 3 is the
# only line selected in the saved list and line 5 is
# the only line selected in the current list
Set current list lMyList
Define list {lCol1}
For lCol1 from 1 to 6 step 1
  Add line to list {lCol1}
End For
Select list line(s) {3}
Save selection for line(s) (All lines)
Deselect list line(s) (All lines)
Select list line(s) {5}
OR selected and saved (All lines)
```

Paste from clipboard

Command group	Flag affected	Reversible	Execute on client
Clipboard	YES	NO	NO

Syntax

Paste from clipboard field-name ([Redraw field][,All windows])

Options

Redraw field	If specified, the command reloads affected window fields with the new value of the data field, after it has performed the operation; note that this takes the 'All windows' option into account
All windows	If specified, the command applies to all open window instances, rather than just the top open window instance

Description

This command pastes the contents of the clipboard into the specified field, current selection or at the insertion point. When the field-name parameter is specified, **Paste from clipboard** pastes the contents of the clipboard into the field replacing the contents of the whole field. However, when the *field-name* parameter is not specified the command will paste the contents of the clipboard at the current selection (a range of selected characters) or the insertion point within the current field.

Example

```
# Copy one field to another then clear the first field
Copy to clipboard iName
Paste from clipboard iDeliveryName (Redraw field)
Clear data iName (Redraw field)
```

POP3Connect

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

POP3Connect (*server*,*username*,*password*[,*stsproc*,*secure* {Default zero insecure;1 secure;2 use STARTTLS},*verify* {Default kTrue}]) **Returns** socket

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

POP3Connect establishes a connection to a POP3 server. If **POP3Connect** succeeds, it returns the socket opened to the POP3 server. You can use this socket with the other POP3 commands which require a socket argument. If an error occurs, **POP3Connect** returns an error code, which is less than zero. Possible error codes are listed in the Web Command Error Codes Appendix.

Note that it is essential that you call POP3Disconnect when you have finished using the connection to the POP3 server.

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. If the server is not using the default POP3 port (110, or 995 for a secure connection), you can optionally append the port number on which the server is listening, using the syntax server:port, for example pop3.mydomain.com:1234.

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

Password is an Omnis character field containing the password for the account specified in the UserName parameter, for example, Secret.

StsProc is an optional parameter containing the name of an Omnis method that **POP3Connect** calls with status messages. **POP3Connect** calls the method with no parameters, and the status information in the variable #S1. The status information logs protocol messages exchanged on the connection to the server.

Secure is an optional Boolean parameter which indicates if a secure connection is required to the server. Pass kTrue for a secure connection, in which case the built-in security technology will be used, so on Windows 'Secure Channel' (Schannel) is used, on macOS 'Secure Transport' is used, and on Linux OpenSSL is used.

POP3Connect also supports an alternative secure option, if you pass secure with the value 2, the connection is initially not secure, but after the initial exchange with the server, **POP3Connect** issues the STLS POP3 command to make the connection secure if the server supports it (see RFC 2595 for details). Authentication occurs after a successful STLS command.

Verify is an optional Boolean parameter which is only significant when Secure is not kFalse. When Verify is kTrue, the command instructs the installed SSL library to verify the server's identity using its certificate; if the verification fails, then the connection will not be established. You can pass Verify as kFalse, to turn off this verification; in this case, the connection will still be encrypted, but there is a chance the server is an impostor. In order to perform the verification, the installed SSL library uses the Certificate Authority Certificates in the cacerts sub-folder of the secure folder in the Omnis folder. If you use your own Certificate Authority to self-sign certificates, you can place its certificate in the cacerts folder, and the installed SSL library will use it after you restart Omnis.

Socket is an Omnis Long integer field which receives the socket for the new connection. If an error occurs, **POP3Connect** returns an error code with a value less than zero. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# Establish a connection to the POP3 server lServer for user
# lUsername using the password lPassword
Calculate lServer as 'my.pop3.server'
Calculate lUserName as 'myusername'
Calculate lPassword as 'mypassword'
POP3Connect (lServer,lUserName,lPassword) Returns iSocket
```

POP3DeleteMessage

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

POP3DeleteMessage (*socket*,*messagenumber*[,*stsproc*]) **Returns** *status*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

POP3DeleteMessage marks a message stored on a POP3 server for deletion. The POP3 server deletes messages marked for deletion when you call POP3Disconnect. Before calling POP3Disconnect, you can call POP3UndoDeletes, to remove the deletion mark from all messages.

Socket is an Omnis Long Integer field containing a socket opened to a POP3 server using POP3Connect.

MessageNumber is an Omnis Long Integer field which identifies the message to be marked for deletion. Message numbers are assigned by the POP3 server, after you call POP3Connect, starting with 1 for the first message, 2 for the second, and so on.

StsProc is an optional parameter containing the name of an Omnis method that **POP3DeleteMessage** calls with status messages. **POP3DeleteMessage** calls the method with no parameters, and the status information in the variable #S1. The status information logs protocol messages exchanged on the connection to the server.

Status is an Omnis Long Integer field which receives the result of executing the command. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# Delete the first message for user lUsername from the POP3
# server lServer
Calculate lServer as 'my.pop3.server'
Calculate lUserName as 'myusername'
Calculate lPassword as 'mypassword'
POP3Connect (lServer,lUserName,lPassword) Returns iSocket
POP3DeleteMessage (iSocket,1)
POP3Disconnect (iSocket) ## message is not deleted until disconnect
```

POP3Disconnect

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

POP3Disconnect (*socket*[,*stsproc*]) **Returns** *status*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

POP3Disconnect closes a connection to a POP3 server.

Socket is an Omnis Long Integer field containing a socket opened to a POP3 server using POP3Connect.

StsProc is an optional parameter containing the name of an Omnis method that **POP3Disconnect** calls with status messages. **POP3Disconnect** calls the method with no parameters, and the status information in the variable #S1. The status information logs protocol messages exchanged on the connection to the server.

Status is an Omnis Long Integer field which receives the result of executing the command. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# Close the connection to the POP3 server lServer
Calculate lServer as 'my.pop3.server'
Calculate lUserName as 'myusername'
Calculate lPassword as 'mypassword'
POP3Connect (lServer,lUserName,lPassword) Returns iSocket
POP3Disconnect (iSocket)
```

POP3ListMessages

Command group	Flag affected	Reversible	Execute on client	P
External commands	YES	NO	NO	A

Syntax

POP3ListMessages (*socket*,*messagenumber*{0 to list all},*list*[,*stsproc*]) **Returns** *status*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

POP3ListMessages lists the messages on a POP3 server. Note that the list excludes any messages marked for deletion (see POP3DeleteMessage You can either list an individual message, or all messages not marked for deletion.

Socket is an Omnis Long Integer field containing a socket opened to a POP3 server using POP3Connect.

MessageNumber is an Omnis Long Integer field which identifies the message to be listed. Message numbers are assigned by the POP3 server, after you call POP3Connect, starting with 1 for the first message, 2 for the second, and so on.

To list all messages, pass 0 as the MessageNumber argument. Otherwise, MessageNumber must identify a single message which is not marked for deletion.

List is an Omnis list field defined to have 2 long integer columns. **POP3ListMessages** clears the list, and then adds a line to the list for each message. Column 1 receives the message number, and column 2 receives the message size in bytes.

StsProc is an optional parameter containing the name of an Omnis method that **POP3ListMessages** calls with status messages. **POP3ListMessages** calls the method with no parameters, and the status information in the variable #S1. The status information logs protocol messages exchanged on the connection to the server.

Status is an Omnis Long Integer field which receives the result of executing the command. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# List details of all messages for user lUserName not marked for
# deletion on the POP3 server lServer in lMailList
Calculate lServer as 'my.pop3.server'
Calculate lUserName as 'myusername'
Calculate lPassword as 'mypassword'
POP3Connect (lServer,lUserName,lPassword) Returns iSocket
Do lMailList.$define(lMsgNum,lMsgSize)
POP3ListMessages (iSocket,0,lMailList) Returns lStatus
```

POP3MessageCount

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

POP3MessageCount (*socket*[,*stsproc*]) **Returns** *messagecount*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

POP3MessageCount returns the count of messages available on a POP3 server. The count includes messages marked for deletion (see POP3DeleteMessage).

Socket is an Omnis Long Integer field containing a socket opened to a POP3 server using POP3Connect.

StsProc is an optional parameter containing the name of an Omnis method that **POP3MessageCount** calls with status messages. **POP3MessageCount** calls the method with no parameters, and the status information in the variable #S1. The status information logs protocol messages exchanged on the connection to the server.

MessageCount is an Omnis Long Integer field which receives the number of messages. If an error occurs, the returned value is an error code with a value less than zero. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# Find out how many messages exist on the POP3 server lServer
# for user lUserName
Calculate lServer as 'my.pop3.server'
Calculate lUserName as 'myusername'
Calculate lPassword as 'mypassword'
POP3Connect (lServer,lUserName,lPassword) Returns iSocket
POP3MessageCount (iSocket) Returns lMessageCount
```

POP3Recv

Command group	Flag affected	Reversible	Execute on client	P
External commands	YES	NO	NO	A

Syntax

POP3Recv (*server*,*user*,*pass*,*list*[,*delete*,*stsproc*,*maxmessages*,*secure* {Default zero insecure;1 secure;2 use STARTTLS},*verify* {Default kTrue}]) **Returns** *status*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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. Possible error codes are listed in the Web Command Error Codes Appendix.

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. If the server is not using the default POP3 port (110, or 995 for a secure connection), you can optionally append the port number on which the server is listening, using the syntax server:port, for example pop3.mydomain.com:1234.

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 binary or 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. Note that

you can pass the message data stored in each row to MailSplit, in order to parse the message. For correct results with many of the encodings supported by MailSplit you must define the list to have a binary column.

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 status messages. **POP3Recv** calls the method with no parameters, and the status information in the variable #S1. The status information logs protocol messages exchanged on the connection to the server.

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.

Secure is an optional Boolean parameter which indicates if a secure connection is required to the server. Pass kTrue for a secure connection, in which case the built-in security technology will be used, so on Windows 'Secure Channel' (Schannel) is used, on macOS 'Secure Transport' is used, and on Linux OpenSSL is used.

POP3Recv also supports an alternative secure option, if you pass secure with the value 2, the connection is initially not secure, but after the initial exchange with the server, **POP3Recv** issues the STLS POP3 command to make the connection secure if the server supports it (see RFC 2595 for details). Authentication occurs after a successful STLS command.

Verify is an optional Boolean parameter which is only significant when Secure is not kFalse. When Verify is kTrue, the command instructs the installed SSL library to verify the server's identity using its certificate; if the verification fails, then the connection will not be established. You can pass Verify as kFalse, to turn off this verification; in this case, the connection will still be encrypted, but there is a chance the server is an impostor. In order to perform the verification, the installed SSL library uses the Certificate Authority Certificates in the cacerts sub-folder of the secure folder in the Omnis folder. If you use your own Certificate Authority to self-sign certificates, you can place its certificate in the cacerts folder, and the installed SSL library will use it after you restart Omnis.

Example

```
# Retrieve all messages for user lUsername from the POP3 server
# lServer and remove the messages from the server
Calculate lServer as 'my.pop3.server'
Calculate lUserName as 'myusername'
Calculate lPassword as 'mypassword'
Calculate lDelete as kTrue
Do lMailList.$define(lMessage)
POP3Recv (lServer,lUserName,lPassword,lMailList,lDelete) Returns lStatus
```

POP3RecvHeaders

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

POP3RecvHeaders (socket,messagenumber,headers[,stsproc]) **Returns** status

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

POP3RecvHeaders reads the message headers for a message stored on a POP3 server. The message must not be marked for deletion.

Socket is an Omnis Long Integer field containing a socket opened to a POP3 server using POP3Connect.

MessageNumber is an Omnis Long Integer field which identifies the message for which the headers are to be read. Message numbers are assigned by the POP3 server, after you call POP3Connect, starting with 1 for the first message, 2 for the second, and so on.

Headers is an Omnis Binary or Character field which receives the headers for the specified message. Note that you can pass the result to MailSplit, in order to parse the headers. When calling MailSplit, pass an empty list variable as the body argument. For correct results with many of the encodings supported by MailSplit you must receive into a Binary field.

StsProc is an optional parameter containing the name of an Omnis method that **POP3RecvHeaders** calls with status messages. **POP3RecvHeaders** calls the method with no parameters, and the status information in the variable #S1. The status information logs protocol messages exchanged on the connection to the server.

Status is an Omnis Long Integer field which receives the result of executing the command. Possible error codes are listed in the Web Command Error Codes Appendix.

Note

POP3RecvHeaders uses an optional POP3 protocol command, so that it may not work with all POP3 servers.

Example

```
# Read the header for the first message stored on the POP3 server
# lServer for user lUserName
Calculate lServer as 'my.pop3.server'
Calculate lUserName as 'myusername'
Calculate lPassword as 'mypassword'
POP3Connect (lServer,lUserName,lPassword) Returns iSocket
POP3RecvHeaders (iSocket,1,lHeaders) Returns lStatus
```

POP3RecvMessage

Command group	Flag affected	Reversible	Execute on client	P
External commands	YES	NO	NO	A

Syntax

POP3RecvMessage (*socket*,*messagenumber*,*message*[,stsproc]) **Returns** *status*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

POP3RecvMessage reads a message stored on a POP3 server. The message must not be marked for deletion.

Socket is an Omnis Long Integer field containing a socket opened to a POP3 server using POP3Connect.

MessageNumber is an Omnis Long Integer field which identifies the message to be read. Message numbers are assigned by the POP3 server, after you call POP3Connect, starting with 1 for the first message, 2 for the second, and so on.

Message is an Omnis Binary or Character field which receives the message. Note that you can pass the result to MailSplit, in order to parse the message. For correct results with many of the encodings supported by MailSplit you must receive into a Binary field.

StsProc is an optional parameter containing the name of an Omnis method that **POP3RecvMessage** calls with status messages. **POP3RecvMessage** calls the method with no parameters, and the status information in the variable #S1. The status information logs protocol messages exchanged on the connection to the server.

Status is an Omnis Long Integer field which receives the result of executing the command. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# Read the first message stored on the POP3 server lServer for user
# lUserName
Calculate lServer as 'my.pop3.server'
Calculate lUserName as 'myusername'
Calculate lPassword as 'mypassword'
POP3Connect (lServer,lUserName,lPassword) Returns iSocket
POP3RecvMessage (iSocket,1,lMessage) Returns lStatus
```

POP3Stat

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

POP3Stat (*server*,*user*,*pass*[,*stsproc*,*secure* {Default zero insecure;1 secure;2 use STARTTLS},*verify* {Default kTrue}]) **Returns** *waiting-messages*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

POP3Stat 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. Possible error codes are listed in the Web Command Error Codes Appendix.

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. If the server is not using the default POP3 port (110, or 995 for a secure connection), you can optionally append the port number on which the server is listening, using the syntax server:port, for example pop3.mydomain.com:1234.

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. **POP3Stat** calls the method with no parameters, and the status information in the variable #S1. The status information logs protocol messages exchanged on the connection to the server.

Secure is an optional Boolean parameter which indicates if a secure connection is required to the server. Pass kTrue for a secure connection, in which case the built-in security technology will be used, so on Windows 'Secure Channel' (Schannel) is used, on macOS 'Secure Transport' is used, and on Linux OpenSSL is used.

POP3Stat also supports an alternative secure option, if you pass secure with the value 2, the connection is initially not secure, but after the initial exchange with the server, **POP3Stat** issues the STLS POP3 command to make the connection secure if the server supports it (see RFC 2595 for details). Authentication occurs after a successful STLS command.

Verify is an optional Boolean parameter which is only significant when Secure is not kFalse. When Verify is kTrue, the command instructs the installed SSL library to verify the server's identity using its certificate; if the verification fails, then the connection will not be established. You can pass Verify as kFalse, to turn off this verification; in this case, the connection will still be encrypted, but there is a chance the server is an impostor. In order to perform the verification, the installed SSL library uses the Certificate Authority Certificates in the cacerts sub-folder of the secure folder in the Omnis folder. If you use your own Certificate Authority to self-sign certificates, you can place its certificate in the cacerts folder, and the installed SSL library will use it after you restart Omnis.

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.

Example

```
# Check to see if there is any e-mail waiting to be received for lUserName
Calculate lServer as 'my.pop3.server'
Calculate lUserName as 'myusername'
Calculate lPassword as 'mypassword'
POP3Stat (lServer,lUserName,lPassword) Returns lWaitingMessages
If lWaitingMessages>0
 # receive mail
End If
```

POP3UndoDeletes

Command group	Flag affected	Reversible	Execute on client	P
External commands	YES	NO	NO	A

Syntax

POP3UndoDeletes (*socket*[,*stsproc*]) **Returns** *status*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

POP3UndoDeletes removes the deletion mark from all messages on the POP3 server marked for deletion.

Socket is an Omnis Long Integer field containing a socket opened to a POP3 server using POP3Connect.

StsProc is an optional parameter containing the name of an Omnis method that **POP3UndoDeletes** calls with status messages. **POP3UndoDeletes** calls the method with no parameters, and the status information in the variable #S1. The status information logs protocol messages exchanged on the connection to the server.

Status is an Omnis Long Integer field which receives the result of executing the command. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# Remove the deletion mark from all messages currently marked for
# deletion on the POP3 Server lServer
Calculate lServer as 'my.pop3.server'
Calculate lUserName as 'myusername'
Calculate lPassword as 'mypassword'
POP3Connect (lServer,lUserName,lPassword) Returns iSocket
POP3UndoDeletes (iSocket) Returns lStatus
```

Popup menu

Command group	Flag affected	Reversible	Execute on client	P
Menus	YES	NO	NO	A

Syntax

Popup menu *menu-name* ([*x-coordinate*,*y-coordinate*,*menu-constructor-parameters*])

Description

This command installs the specified menu as a popup menu at the specified location. The location is the x,y screen coordinate relative to the (0,0) position. Under Windows, the coordinate (0,0) is the point directly under the menu bar within the Omnis application window. Under Linux and macOS, (0,0) is literally the top left corner of the screen. If you omit the x,y coordinates the menu pops up at the current mouse/pointer position.

The mouseover() function returns the mouse/pointer position relative to the open window and not the Omnis application window. Using this function to specify the x and y position of the popup menu may not produce the effect you want.

Popup menu behaves much like Popup menu from list except the source of the popup is a user-defined menu. It clears the flag if the user does not select a menu line, otherwise, the line selected from the popup is executed.

If desired, you can supply parameters after the x and y coordinates. The command passes these parameters to the constructor of the user-defined menu instance. When passing constructor parameters, leave the x and y coordinate parameters as either empty or #NULL to popup the menu in the current mouse position.

Example

```
# Prevent the default context menu appearing and open the menu mView instead
# $event of object on window class
On evRMouseDown ## requires the property $rmouseevents set to kTrue
  Process event and continue (Discard event)
  Popup menu mView
```

Popup menu from list

Command group	Flag affected	Reversible	Execute on client
Menus	YES	NO	NO

Syntax

Popup menu from list *list-name* **at** *x-coordinate, y-coordinate*

Description

This command installs the specified list as a popup menu at the specified x,y screen location. Under Windows, the coordinate 0,0 is the point directly under the menu bar within the application area. Under Linux and macOS, 0,0 is literally the top left corner of the screen. If you omit the x,y coordinate from this command the menu pops up at the current mouse/pointer position.

Popup menu from list behaves much like Popup menu except the source of the menu is a list. The specified list can contain any number of rows but only the first column and a limited number of rows are displayed in the popup menu.

This command clears the flag if the user does not select a list line and LIST.$line is unaffected. After the command has executed you can use lst() to return the line selected.

The mouseover() function returns the mouse/pointer position relative to the open window and not the Omnis application window. Using this function to specify the x and y position of the popup menu may not produce the effect you want.

Example

```
# Prevent the default context menu appearing and open a menu containing the lines defined in the list
# lMenuLines
# $event of object on window class
On evRMouseDown ## requires the property $rmouseevents set to kTrue
  Process event and continue (Discard event)
  Do lMenuLines.$define(lMenuLine)
  Do lMenuLines.$add('Option 1')
  Do lMenuLines.$add('Option 2')
  Do lMenuLines.$add('Option 3')
  Popup menu from list lMenuLines at
```

Prepare for edit

Command group	Flag affected	Reversible	Execute on client	P
Changing data	YES	YES	NO	A

Syntax

Prepare for edit

Description

This command prepares Omnis for editing data. It brings records into memory ready for updating and rereads the current records when in multi-user mode in case another user has made a change to a record since it was read in. Your method can then alter the values of the records. The contents of the current record buffer are not written back to disk until Update files is encountered.

If there is a window open and you require data to be entered via that window, Enter data is required after the **Prepare for edit**.

Prepare for edit/insert mode is cleared only by a Cancel prepare for update, Update files or Quit all methods command. You can build lists, print reports and change the main file in the middle of an update without cancelling the Prepare for... mode.

Multi-user considerations

Records in the current record buffer from Read/write files will be locked when **Prepare for edit** is executed, so as to prevent simultaneous editing of a record. The lock is removed by Update files or any command which cancels the Prepare for mode.

If Wait for semaphores is active, a **Prepare for edit** will wait for a record to become available if another workstation has locked it. If the user presses Ctrl-Break/Ctrl-C/Cmnd-period while waiting for access, the command fails and processing halts. With Do not wait for semaphores active, a record lock returns control to the method with the flag false.

In the following method, the Edit mode is used to process the whole of a file. Enter data is not used as no user intervention is required. Update files writes data to the disk and clears the Prepare for.. mode and record locks.

Example

```
# The following example is equivalent to the 'edit record' on the commands menu which can be installed using '
Prepare for edit
Enter data
If flag true
  Update files
Else
  Clear main & con
  Redraw {wMyWindow}
End If
# In the following method, the Edit mode is used to process the whole of a file. Enter data
# is not used as no user intervention is required. Update files writes data to the disk and
# clears the Prepare for.. mode and record locks.
# In ‘Wait for semaphores’ mode:
Set main file {fAccounts}
Find first on fAccounts.Code
While flag true
  Prepare for edit
  Calculate fAccounts.Balance as fAccounts.Balance-10
  Update files
  Next
End While
# In ‘Do not wait for semaphores’ mode:
Set main file {fAccounts}
Find first on fAccounts.Code
While flag true
  Repeat
```

```
    Prepare for edit
  Until flag true
  Calculate fAccounts.Balance as fAccounts.Balance-10
  Repeat
    Update files
  Until flag true
  Next
End While
# In the next Edit example, the Enter data command is included in the method so that the user
# can edit the record from the keyboard. Again, the command Update files cancels the Prepare
# for update mode and writes data to the disk.
Prepare for edit
Enter data
Update files if flag set
# The next example has been written to control record locking by preventing Omnis from waiting
# for a record lock. It takes the form of general purpose 'prepare for edit' which you can
# call with a number which tells it how many times to try for a lock if the record is locked by
# another user.
# general Prepare for edit
# declare Parameter lTries (Number Long Integer)
Do not wait for semaphores
Calculate lCount as 1
Repeat
  Prepare for edit
  Calculate lCount as lCount+1
Until #F|(lCount>lTries)
# Keeps trying until flag true OR counter>TRIES
Wait for semaphores
```

Prepare for export to file

Command group	Flag affected	Reversible	Execute on client
Importing and Exporting	YES	NO	NO

Syntax

Prepare for export to file {*export-format*}

Export Formats
Delimited (commas)
Delimited (tabs)
One field per line
Omnis data transfer
Delimited (user delimiter)

Description

This command prepares to export records to a file in one of the specified data formats. The file must previously have been set using Set print or export file name.

Example

```
# export to a file called myExport.txt in the root of your omnis tree
Calculate lExportPath as con(sys(115),'myExport.txt')
Set print or export file name {[lExportPath]}
Prepare for export to file {Delimited (commas)}
```

```
Export data lExportList
End export
Close print or export file
```

Prepare for export to port

Command group	Flag affected	Reversible	Execute on client
Importing and Exporting	YES	NO	NO

Syntax

Prepare for export to port {*export-format*}

Export Formats
Delimited (commas)
Delimited (tabs)
One field per line
Omnis data transfer
Delimited (user delimiter)

Description

This command prepares to export records to a port in one of the specified data formats. The file must previously have been set using Set port name or Prompt for port name.

Example

```
# export to port Com1
Set port name {COM1:}
Prepare for export to port {Delimited (commas)}
Export data lExportList
End export
```

Prepare for import from client

Command group	Flag affected	Reversible	Execute on client	P
Importing and Exporting	YES	NO	NO	V

Syntax

Prepare for import from client {*export-format*}

Export Format
Delimited (commas)
Delimited (tabs)
One field per line
Omnis data transfer
Delimited (user delimiter)

Description

Example

```
Prepare for import from client {Delimited (commas)}
If flag true
  Import data lImportList
End If
End import
```

Prepare for import from file

Command group	Flag affected	Reversible	Execute on client	P
Importing and Exporting	YES	NO	NO	A

Syntax

Prepare for import from file {*export-format*}

Export Formats
Delimited (commas)
Delimited (tabs)
One field per line
Omnis data transfer
Delimited (user delimiter)

Description

This command prepares Omnis for a series of Import data commands. You must specify the format for the import data as the parameter, otherwise an error will occur. The parameter can contain square bracket notation but must evaluate to a valid import format name. You should use the Set import file name command to specify the name of the file to be read in.

If the data matches the specified import format, the flag is set. However, if the data does not match the import format, the flag is cleared.

When data is imported via a method rather than the **Utilities** menu, you must open a window which defines the fields in which the incoming data must be placed. The example below shows a typical import data method.

You can use a $control() method in conjunction with the Import data command.

If there are too few fields on the window, imported fields will be lost. If there are too many, the extra fields are cleared. You can use the Do not flush command to speed up the import when there is only one user logged into the data file.

Example

```
# import from a csv file called myImport.txt in the root of your Omnis tree
Calculate lImportPath as con(sys(115),'myImport.txt')
Set import file name {[lImportPath]}
Prepare for import from file Delimited (commas)
Import data lImportList
End import
Close import file
```

Prepare for import from port

Command group	Flag affected	Reversible	Execute on client
Importing and Exporting	YES	NO	NO

Syntax

Prepare for import from port {*export-format*}

Export Formats
Delimited (commas)
Delimited (tabs)
One field per line
Omnis data transfer
Delimited (user delimiter)

Description

This command prepares Omnis for importing data from a port. It is similar to the Prepare for import from file command. The user can cancel the import of data while **Prepare for import from port** is waiting for data from the port. If this happens, Omnis clears the flag.

Set port name defines which port is used. Under macOS, the choice is 1 (Modem port) or 2 (Printer port). Under Linux and Windows, the choices are Com1:, Com2:, and so on.

Example

```
Set port name {COM1:}
Prepare for import from port {One field per line}
Repeat
  Import field from file int lImportField
Until lImportField='start data'
Do method ImportData
Close import file
```

Prepare for insert

Command group	Flag affected	Reversible	Execute c
Changing data	YES	YES	NO

Syntax

Prepare for insert

Description

This command prepares Omnis for inserting new data into the main file. It clears the main file and prepares to insert a new record into the main file. All Read/write non-main file records in the current record buffer are reread if a record has been changed. You can edit data in all the read/write files in the buffer, other than the main file.

The Enter data command is required only if the user is to enter data via a window. Data is not written to the disk until Update files is executed.

Prepare for edit/insert mode is cleared only by a Cancel prepare for update, Update files or a Quit all methods command. You can build lists, print reports and change the main file in the middle of an insert without canceling the Prepare for... mode.

If the main file is changed while in Prepare for insert mode, the main file at the time of the **Prepare for insert** is used when Update files is encountered.

In multi-user mode, the Prepare for... commands reread the current records from the data file if another user has edited a record.

Example

```
# The following example is equivalent to the 'insert record' on the commands menu which can be
# installed using 'Install menu *Commands'
Prepare for insert
Enter data
If flag true
  Update files
Else
  Clear main & con
  Redraw {wMyWindow}
End If
```

Prepare for insert with current values

Command group	Flag affected	Reversible	Execute c
Changing data	YES	YES	NO

Syntax

Prepare for insert with current values

Description

This command prepares Omnis for inserting new data into the main file using the values in the current record buffer as a starting point. **Prepare for insert with current values** differs from Prepare for insert in that the fields in the main file are not cleared.

In multi-user mode, the Prepare for... commands reread the current records from the data file if another user has edited a record.

Example

```
Set main file {fAccounts}
Prepare for insert with current values
If flag false
  Quit method kFalse
End If
Enter data
Update files if flag set
```

Prepare for print

Command group	Flag affected	Reversible
Reports and Printing	YES	YES

Syntax

Prepare for print ([*Ask for job setup*][,*Do not finish others*]) {*instance-name* (*parameters*)}

Options

Ask for job setup	If specified, the command opens the job setup dialog
Do not finish others	If not specified, all reports in progress are terminated before the new report is started

Description

This command prepares Omnis for record-by-record report printing. You specify the report instance name and you can add a list of $construct parameters for the report instance. The default instance name is the name of the report class itself.

You must put **Prepare for print** after any Set report name, Select destination..., Set port name, Set print or export file name, Set sort field and Report parameter commands and before the first Print record command.

The *Ask for job setup* option opens the job setup dialog that lets you select the number of copies, paper trays, the printer, and so on, for the current print job.

Prepare for print has the *Do not finish others* option which when checked allows multiple reports to be in progress at the same time. If this is unchecked (the default) all reports in progress are terminated before the new report is started, which is compatible with earlier versions of Omnis.

The flag is set if the command is successful, errors cause a message to be displayed. If placed in a reversible block, the **Prepare for print** mode is canceled and the totals printed when the command is reversed.

All the Print commands give an error if no report is selected, or if the report is printed to a port and no port is selected.

When reports are printed record-by-record using Print record in a loop, the sort fields set up in the report class still trigger the subtotals. No sorting takes place and, therefore, you must take care in the choice of index. You can trigger subtotals from the method by including a variable on the first line of the report class, including it in the sort fields and then using the method to change its value when required.

The **Prepare for print** mode is terminated or cancelled by End print. You must include an End print after a **Prepare for print** even if a totals section is not required.

Example

```
# Print report record by record
Prompt for destination
If flag true
  Set main file {fAccounts}
  Set report name rMyReport
  Send to screen
  Prepare for print
  Find first on fAccounts.Code
  While flag true
    # tVar1 is a sort field placed on line 1 of the report
    # class used to trigger subtotal section 1
    Calculate tVar1 as fAccounts.Surname
    Print record
    Next
  End While
  End print
End If
```

Previous

Command group	Flag affected	Reversible	Execute on
Finding data	YES	YES	NO

Syntax

Previous on *field-name* ([*Exact match*][,*Use search*])

Options

Exact match	If specified, the index value of the field in suitable records must equal the current value
Use search	If specified, the command uses the current search to select data

Description

This command locates the previous record using the current find table. The **Previous** command works in the same way as the corresponding option on the Commands menu but with no redraw, allowing you to work through a file. It is usually used after a Find command which creates a find table of records.

If the Index field, *Exact match* and/or *Search* option used in the Next is incompatible with the preceding Find, a new table is built. Normally, the parameters in this command are left blank so that the current find table is used.

If the **Previous** command does not follow a Find, a find table is built for the current main file before doing the **Previous**.

If an indexed field is specified, **Previous** on SU_NAME for example, the find table is just the index order for the field. The *Use search* option creates a table for the current main file in which the search specification is implicitly stored. Thus, changes to the search do not affect the find table once it is created.

Once the previous record is located, the main and connected files are read into the current record buffer and the flag is set, otherwise, the flag is cleared. An error occurs whenever **Previous** on FIELD is performed on a non-indexed field.

If the *Exact match* option is chosen, the previous record with the same index value is found, or the flag is cleared if no previous records exist with the same index value.

If you use **Previous** with a search, it finds the **previous** record listed on the index table which meets the search criteria.

Example

```
# Find records in descending order
Find last on fAccounts.Code
While flag true
  OK message {Found account [fAccounts.Code]}
  Update files
  Previous
End While
```

Print check data log

Command group	Flag affected	Reversible	Ex
Data management	NO	NO	NO

Syntax

Print check data log

Description

This command prints the current contents of the check data log to the current report destination. There is no need for the log to be open.

Example

```
Check data (Check indexes) ## all files
If flag true
  Print check data log
Else
  OK message {Check data only works if one user is logged on}
End If
```

Print class

Command group	Flag affected	Reversible	Execute on client
Classes	NO	NO	NO

Syntax

Print class {*class-name*}

Description

This command prints the field list and methods (if any) for the specified class. The example prints the field list and/or methods for all the classes in the current library.

Example

```
# generate list of all classes in the current library
Calculate iList as $clib.$classes.$makelist($ref.$name)
Do iList.$redefine(iClassName)
# loop through the list and print the results
For lNum from 1 to iList.$linecount step 1
  Do iList.[lNum].$loadcols()
  Print class {[iClassName]}
End For
```

Print record

Command group	Flag affected	Reversible
Reports and Printing	YES	NO

Syntax

Print record {*instance-name*}

Description

This command prints a single record of the specified report instance. You use it when printing a report on a record-by-record basis and usually within a loop. It provides greater control over the report generator than Print report. If you omit the report instance name **Print record** is applied to the most recently started report instance ($ireports.$first).

Each time **Print record** is encountered, a record section of the report is printed to the selected output using the data in the CRB. Any page heading, subtotal heading and subtotal sections before the record section are printed where necessary.

Subtotal sections are printed whenever the sort fields change value, provided that the fields entered in the Sort Fields dialog have Subtotals set to True.

The flag is cleared if:

- no Prepare for print is used, or
- the user cancels the report by pressing Ctrl-Break/Ctrl-C/Cmnd-period, or
- there is an error.

These errors will not cause Omnis to execute a Quit all methods. If the flag is cleared, Omnis will not execute any further **Print record** commands until it encounters another Prepare for print.

Example

```
# Print report record by record
Set main file {fAccounts}
Set report name rMyReport
Send to screen
Prepare for print
Find first on fAccounts.Code
While flag true
  Print record
  Next
End While
End print
```

Print report

Command group	Flag affected	Reversible
Reports and Printing	YES	NO

Syntax

Print report ([*Ask for job setup*][,*Use search*][,*Do not finish others*]) {*instance-name* (parameters)}

Options

Ask for job setup	If specified, the command opens the job setup dialog
Use search	If specified, the command uses the current search to select data
Do not finish others	If not specified, all reports in progress are terminated before the new report is started

Description

This command prints the specified report instance to the selected output. You specify the report instance name and you can add a list of $construct parameters for the report instance. The default instance name is the name of the report class itself.

Subtotal sections are printed whenever the sort fields change value, provided that the fields entered in the Sort Fields dialog have Subtotals set to True.

You specify sort fields and the main file or list as part of the report parameters. If the main file has not been set in the report class, the current main file is used. You can override all the parameters in the class using the appropriate commands, for example, Set left margin.

Print report does not use the current record buffer but a special memory buffer to load in and sort records. Thus **Print report** does not affect Prepare for mode and does not lose current records. If the report is printed from a list, data is read directly from the report main list, as specified in the parameters for the report. LIST.$line is unaffected.

The *Ask for job setup* option opens the job setup dialog that lets you select the number of copies, paper trays, the printer, and so on, for the current print job.

All records are printed unless the *Use search* option is specified. In this case, only the records matching the current search class are printed. It is not necessary to use Prepare for print before **Print report**.

The *Do not finish others* option allows multiple reports to be in progress at the same time. If this is unchecked (the default) all reports in progress are terminated before the new report is started, which is compatible with earlier versions of Omnis.

The flag is cleared if the report is cancelled before completion by the user or in the event of an error. Most errors will display a message but will not cause Omnis to Quit all methods.

Example

```
# Print the report rMyReport
Set report main file {fAccounts}
Set report name rMyReport
Clear sort fields
Set sort field fAccounts.Surname
Send to screen
Print report
```

Print report from disk

Command group	Flag affected	Reversible
Reports and Printing	YES	NO

Syntax

Print report from disk {*file-name*}

Description

This command prints the contents of the specified disk file to the current report destination. The specified file must contain output generated using the Disk printing device.

Example

```
# Print the report rMyReport to disk and then print
# the report from disk to screen
Calculate lFileName as con(sys(115),'myreport.rep')
Set report name rMyReport
Do $cdevice.$assign(kDevDisk)
Do $prefs.$reportfile.$assign(lFileName)
Print report
Close print or export file
Send to screen
Print report from disk {[lFileName]}
```

Print report from memory

Command group	Flag affected	Reversible
Reports and Printing	YES	NO

Syntax

Print report from memory *field-name*

Description

This command prints the contents of the specified binary field or variable to the current report destination. The specified field or variable must contain output generated using the Memory printing device.

Example

```
# Print the report rMyReport to memory and then print
# the report from memory to screen
Set report name rMyReport
Do $cdevice.$assign(kDevMemory)
Do $prefs.$reportdataname.$assign(iBinVar)
Print report
Send to screen
Print report from memory iBinVar
```

Print top window

Command group	Flag affected	Reversible	Execute on client
Windows	YES	NO	NO

Syntax

Print top window

Description

This command prints the top window to the current print destination.

Print top window scales the image if it is too big for the paper (or page preview). You can disable scaling by adding the "disablePrint-TopWindowScaling" item to the "ide" section of the config.json file and setting it to true (if omitted, the setting is false by default and scaling will occur).

Example

```
# Print the current window to screen
Send to page preview
Print top window
```

Print trace log

Command group	Flag affected	Reversible	Execute on client
Debugger	NO	NO	NO

Syntax

Print trace log

Description

This command prints the contents of the trace log to the current print destination.

Process event and continue

Command group	Flag affected	Reversible	Execute on client
Events	NO	NO	NO

Syntax

Process event and continue ([*Discard event*])

Options

Discard event	If specified, Omnis discards the active event (meaning that no further processing will occur for that event)

Description

This command causes the current event to be processed immediately allowing the event handler method containing the command to continue to execute. Normally, the default processing for an event takes place when all the event handler methods dealing with the event have finished executing. It is not possible to have active unprocessed events when waiting for user input so the default processing is carried out for any active events after an Enter data command has been executed or at a debugger break. Therefore if required, you can use this command to override the default behavior and force events to be processed allowing the event handler method to continue.

The *Discard event* option lets you discard the active event.

Example

```
# This code would cause the OK event to be thrown away before the Enter Data starts
On evOK
  Process event and continue (Discard event)
  Open window instance wMyWindow
  Enter data
```

Prompt for data file

Command group	Flag affected	Reversible	Execute on client
Data files	YES	NO	NO

Syntax

Prompt for data file ([*Do not close other data*][,*Read-only*][,*No conversion by runtime*][,*Convert without user prompts*][,*Full Unicode conversion*]) {[*internal-name*] or odb://[address:port][,internal-name]}

Options

Do not close other data	If specified, the command does not close all open data files before opening the specified data file
Read-only	If specified, the data file is opened in read-only mode
No conversion by runtime	Omnis normally offers to convert data files created by an earlier version of Omnis. If this option is specified, the runtime version of Omnis will not offer to convert the file, and the command will fail
Convert without user prompts	If specified, and conversion is allowed, Omnis will immediately perform the conversion without giving the user any prompts that require a response; also, the user cannot cancel the conversion

Full Unicode conversion	Unicode Studio only. If specified, and convert without user prompts is specified, do full Unicode conversion instead of quick conversion (quick conversion is only ok when you know all character data in the file is 7 bit)

Description

This command prompts the user to enter the name of a data file. A dialog box is displayed that lets the user choose a data file. An error message e.g. "Unable to find data file" is generated if the selected file cannot be opened, and the user is forced to select another file name or Cancel. If the user selects Cancel, the flag is cleared and the original data file remains selected.

The selected file is opened in shared mode unless the volume does not support record locking.

The existing open data files remain open if the *Do not close other data* option is selected. In this case, the new data file becomes the "current" data file and this becomes the default data file for file classes which have not been associated with a particular data file using the Set default data file command. If the *Do not close other data* option is not specified, all other open data files are closed even if the command fails.

If an attempt is made to open a data file which is already open, that data file is closed and reopened. The *Read-only* Studio/Omnis 7 check box causes the data file to be opened in read-only mode. This lets you open an Omnis 7 data file in read-only mode in Omnis Studio without conversion taking place.

If you select the *No conversion by runtime* option, and the data file was created with a previous version of Omnis, then the runtime version of Omnis will not convert the data file. The default is that an Omnis runtime will ask the user if they want to convert the data file.

If the data file is to be accessed using the ODB (Omnis Data Bridge), then you indicate this using a special syntax:

```
odb://[address:port]
```

where address:port is the TCP/IP address and port number of the ODB server, e.g. 127.0.0.1:5900. Omnis opens a dialog that allows you to select a data file handled by the ODB server. You can omit address:port, in which case Omnis uses the address and port stored in the $odbserver root preference. Note that the value of $odbserver is stored in the file odb.txt in the studio folder of the Omnis installation tree.

Example

```
Test if file exists {Orders.df1}
If flag true
  Open data file {Orders.df1}
Else
  Prompt for data file
  If flag false
    Quit method
  End If
End If

# Example 2 - Prompt for a data file on a specific ODB server
Prompt for data file {odb://127.0.0.1:5900}
# Example 3 - Prompt for a data file using the ODB server identified by $prefs.$odbserver
Prompt for data file {odb://}
```

Prompt for destination

Command group	Flag affected	Reversible
Report destinations	YES	YES

Syntax

Prompt for destination

Description

This command displays the report destination window so that the user can select the destination for the report. The user can choose from destinations including: printer, screen, page preview, file, port, and clipboard.

If the command is part of a reversible block, the destination reverts to its former identity when the method terminates. If the user selects the Cancel button on the dialog, the flag is cleared.

Example

```
# allow user choice to where to print report to
Set report name rMyReport
Prompt for destination
If flag true
  Print report
End If
```

Prompt for import file

Command group	Flag affected	Reversible	Execute on client
Importing and Exporting	YES	YES	NO

Syntax

Prompt for import file

Description

This command prompts the user to select the name of the import file. The flag is set if the import file is successfully selected, otherwise a Cancel clears the flag, closes the current file and closes the dialog. You use the selected file in any subsequent Import data commands.

If you use **Prompt for import file** in a reversible block, the import file is closed when the method containing the reversible block terminates.

Example

```
Prompt for import file
Prepare for import from file {Delimited (commas)}
Import data lImportList
End import
Close import file
```

Prompt for input

Command group	Flag affected	Reversible	Execute c
Message boxes	YES	NO	NO

Syntax

Prompt for input *prompt/title/icon-id/max-chars* **Returns** *return-value* ([*Sound bell*][,*Cancel button*][,*Upper case only*][,*Password entry*][,*Prompt above entry*])

Options

Sound bell	If specified, the system bell sounds when the command displays the message
Cancel button	If specified, the message has a cancel button
Upper case only	If specified, all input is converted to upper case at the user interface
Password entry	If specified, all input is displayed as '*' or a solid circle at the user interface
Prompt above entry	If specified, the prompt is displayed above the entry field rather than the default which is to the left of the entry field

Description

This command opens a message box requesting a value from the user. You can specify the text for the prompt, title and icon for the message box, and the maximum number of characters for the input. If the user enters a value and presses OK, the command sets the flag and returns the user value. The command is not reversible.

The first parameter for the **Prompt for input** command is the prompt-text which is the prompt displayed to the left of the entry field by default; you can place the prompt text above the entry field using the *Prompt above entry* option. You can also enter a title for the message box. The prompt and title default to empty. Note that if you want to enter an empty title, you need to enter '/ /' to avoid ambiguity with the newline convention.

You can specify an icon for the message box using the icon-id of an icon from the OmnisPic or UserPic icon data file. Zero is the default which means no icon. You can use one of the icon size constants enclosed in square brackets with the icon id to specify a non-default size, for example, [1710+k48x48]. You can specify the maximum number of characters that the user can enter in max-chars. This defaults to the maximum length defined in your return field. The return-field can specify an initial value for the entry field on the message box, and receives the value entered after the user clicks OK.

The *Sound bell* option causes the system beep to sound when the message box opens. The *Cancel button* option adds a Cancel button to the message box. The flag returns false if the user presses the Cancel button. The *Upper case* only option forces all input to be upper case, while the *Password entry* option hides the input, by displaying '*' for each character entered.

Example

```
# Prompt for a username and greet the user
Prompt for input Please enter your nam Returns lUserName (Sound bell ,Cancel button,Prompt above entry)
If len(lUserName)
  OK message (Icon) {Hello [lUserName]}
End If
```

Prompt for library

Command group	Flag affected	Reversible	Execute on client
Libraries	YES	NO	NO

Syntax

Prompt for library ([*Do not close others*][,*Enable conversion by runtime*][,*Do not open startup task*][,*Convert without user prompts*]) {*internal-name* (*startup-parameters*)}

Options

Do not close others	If specified, the command does not close all open libraries before opening the specified library

Enable conversion by runtime	The development version of Omnis offers to convert libraries created by an earlier version of Omnis. If this option is specified, the runtime version of Omnis will also offer to convert such libraries
Do not open startup task	If specified, the command does not construct an instance of the startup task when it opens the library
Convert without user prompts	If specified, and conversion is allowed, Omnis will immediately perform the conversion without giving the user any prompts that require a response; also, the user cannot cancel the conversion

Description

This command prompts the user for a library file. You can specify the *internal name* and startup task construct *parameters* of the library to be opened, together with the *Do not close others*, *Do not open startup task*, and *Enable conversion by runtime* options.

If the *internal name* of an opened library is specified, a check is made to ensure the internal name is unique among the open libraries; a runtime error occurs if this is not the case. If no internal name is specified, the default internal name is the disk name of the file with the path name and suffix removed. For example, the internal name for 'hd:myfiles:testlib.lbs' is 'testlib'.

If an attempt is made to open a library which is already open, that library is closed and reopened. Refer to Close Library for the consequences of closing a library. If the user cancels the Select Library dialog, the flag is cleared and no libraries are closed.

Do not close others

The *Do not close others* option lets you keep open all other libraries. If the Do not close others option is not selected, then all other open libraries are closed when the user opens a new library, including the one containing the currently executing method.

Passwords

If the library does not need a master password, it is opened at the master level, otherwise the usual prompt for password dialog is opened. The library is closed and a flag false returned if this dialog is closed without a password being entered.

Startup task

If the *Do not open startup task* option is specified, the startup task construct for the opened library is not called and there is no startup task instance. Otherwise, the startup task $construct() method is called and the parameters for it are passed.

Enable conversion by runtime

If you select the *Enable conversion by runtime* option, and the library was created with a previous version of Omnis, then the runtime version of Omnis can convert the library if the user allows. The default is that an Omnis runtime will not ask the user if they want to convert the library.

Example

```
# Prompt the user for a library path and open the selected
# library
Prompt for library (Do not close others)
If flag true
  OK message {Library Opened!}
End If
```

Prompt for page setup

Command group	Flag affected	Reversible
Report parameters	YES	NO

Syntax

Prompt for page setup

Description

This command displays the Printer page setup dialog box. This dialog allows the page size, orientation and printer's effects to be chosen before a report is printed. The flag is set if the dialog is closed by clicking on the OK pushbutton. Cancel clears the flag and leaves the page parameters unchanged.

Example

```
# Prompt for page setup before printing the
# report rMyReport
Set report name rMyReport
Prompt for page setup
If flag true
  Print report
End If
```

Prompt for port name

Command group	Flag affected	Reversible
Report destinations	YES	YES

Syntax

Prompt for port name

Description

This command displays the Set port dialog box that lets the user select a port. The flag is set if the port is successfully selected; if the user cancels, the flag is cleared and the port closed.

You can set the baud rate and other parameters for the port using Set port parameters.

If the command is in a reversible block, the port is closed when the method terminates.

Example

```
Prompt for port name
Prepare for import from port
If flag true
  Import data lImportList
End If
End import
Close port
```

Prompt for print or export file

Command group	Flag affected	Reversible
Report destinations	YES	YES

Syntax

Prompt for print or export file

Description

This command displays the Select Print or Export File dialog. The flag is set if the file is successfully selected. If the file exists already, a further dialog lets you delete it. If the user cancels, the flag is cleared and the file is closed.

If the command is in a reversible block, the file is closed when the method terminates.

Example

```
Prompt for print or export file
If flag true
  Send to file
  Set report name rMyReport
End If
Print report
```

Prompted find

Command group	Flag affected	Reversible	Execute on
Finding data	YES	YES	NO

Syntax

Prompted find ([*Exact match*])

Options

Exact match	If specified, the index value of the field in suitable records must equal the current value

Description

This command prompts the user to enter a value in an indexed field on the current window and locates the record which most closely matches that value. The user can use the Tab key to select an indexed field. The Find field is the current field for the window when the user clicks on the OK button.

Once the user enters a value in the Find field and clicks OK, Omnis locates the record most closely matching this value, the main and connected files are read into the current record buffer and the flag is set. If the indexed field is in a connected file, the find continues until a record connected to a valid main file record is located. The current index, as used by Next and Previous, is set to the Find field.

If the exact field value cannot be matched, the next highest value in the index is located. You use the *Exact match* option if you want only the exact match.

Example

```
# Find the record for an indexed value entered into
# the current window instance
Prompted find
If flag true
  Do $cinst.$redraw()
End If
```

Put file name

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

Put file name (*path* [,*dialog-title*] [,*prompt*] [,*default*]) **Returns** *err-code*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This command opens the standard Save as... dialog for the current Operating System, in order to obtain the path of a file from the user. You would typically use this command to prompt the user for the path of a new file. If you want to prompt the user to enter the path of an existing file, use the Get file name command instead.

You can pass a title for the dialog, in dialog-title. You can specify a default value for the entry field in default.

Put file name returns the full pathname of the file the user entered in path, or empty if no file was entered (that is, the Cancel button was clicked). The named file is not opened or created.

It returns an error code (See Error Codes), or zero if no error occurs.

Example

```
# prompt the user for a file name, default to myfile.txt
Put file name (lPathname,'select a file','','myfile.txt') Returns lErrCode
If len(lPathname)=0
  # cancel button pressed
Else
  # file name entered
End If
```

Queue bring to top

Command group	Flag affected	Reversible	Execute on client
Events	NO	NO	NO

Syntax

Queue bring to top window-instance-name

Description

This command queues a “bring to top” event for the specified window instance as if the user had clicked on the window instance with the mouse. The command brings the window instance to the fore and generates evWindowClick and evToTop events. If, at runtime, the specified window instance does not exist, the command will do nothing.

Example

```
Open window instance wMyWindow/wInst1/STK
Open window instance wMyWindow/wInst2/STK
Queue bring to top wInst1 ;; brings the instance wInst1 to the top
```

Queue cancel

Command group	Flag affected	Reversible	Execute on client
Events	NO	NO	NO

Syntax

Queue cancel

Description

This command queues a “cancel” event, as if the user had clicked on the Cancel button or pressed the escape key. The command takes no parameters.

Example

```
# Setup a timer call to cancel the edit after 120 seconds
Set timer method sec cGeneral/TimerSetup
Prepare for edit
Enter data
Update files if flag set
# Code for cGeneral/TimerSetup
Send to trace log {No edit has occurred - Timed out}
Queue cancel
```

Queue click

Command group	Flag affected	Reversible	Execute on client
Events	NO	NO	NO

Syntax

Queue click ([*Shift*][,*Command/Ctrl*]) {*field-name* (*selection-range*)}

Options

Shift	If specified, the queued event behaves as if the shift key has been pressed
Command/Ctrl	If specified, the queued event behaves as if the command/ctrl key has been pressed

Description

This command queues a “mouse click” event on a specified field, that is, it simulates a user-generated mouse click/drag operation on a field. You must specify the name of the field as a parameter, including the click positions within the field (that is, Start Row, Finish Row for lists and Start Character, Finish Character for text field selection). The specified field will get the focus.

There are options for including up to three modifier keys (that is, *Shift*, *Ctrl/Cmnd*) along with the click.

The *field name* parameter must be the name of a window field, not the name of the method associated with the field or the data name ($name, not $dataname).

Queue clicks returns an error if the field cannot be found. You can turn off this error by setting the option “reportQueueCommand-FieldNotFoundErrors” to false, located in the “defaults” section of config.json.

Queue click for pushbuttons

If the specified field is a pushbutton it is activated and an evClick event is generated as if the user had clicked on the button.

Queue click for Radio buttons and check boxes

If the specified field is a check box or set of radio buttons, the check box field or group of radio buttons is checked/unchecked accordingly, and an evClick event is generated. Methods behind radio buttons and check boxes run as if the user had clicked on the window fields.

Queue click for Radio groups

If the specified field is a radio group, you identify the member of the group that is to receive the click by setting the *selection-range* parameter to the value of$dataname that corresponds to the member. When the command executes, Omnis checks the member and generates an evClick event. Event methods for the radio group run as if the user had clicked on the field.

Queue click on Edit fields

You can specify a range of characters. For example, the parameter field-name (2,5), highlights the characters within cursor positions 2 to 5 (that is, characters 3 to 5). Note that cursor position 0 is to the left of character 1, and cursor position 1 is to the right of character 1 (or to the left of character 2).

If Shift is selected and 5 is passed as the selection point, all characters between the current cursor position and cursor position 5 will be highlighted.

As the **Queue click** examples for Edit show, the two parameters act as a “click on, drag to” key operation.

Queue click for lists

If the specified field is a window list box or grid, the range is interpreted as a range of list lines. For example, the parameter list-field-name (2,5), selects the lines 2 to 5 (if $multipleselect for the list field is set), and the current line will be set to 2. An evClick event is generated after the specified lines have been selected.

Example

```
# Queue click for edit fields

# highlight characters 3 to 7
Queue click {myEntryField (7,2)}

# highlight characters 6 to 9
Queue click {myEntryField (5,7)}

# assuming the current cursor is at position 15,
# characters 9 to 15 are highlighted
Queue click (Shift) {myEntryField (8)}

# assuming the current cursor is at position 15,
# characters 16 to 22 are highlighted
Queue click (Shift) {myEntryField (22)}

# assuming the current cursor is at position 15,
# characters 10 to 15 are highlighted
Queue click (Shift) {myEntryField (7,9)}

# assuming the current cursor is at position 15,
# characters 8 to 15 are highlighted
Queue click (Shift) {myEntryField (9,7)}

# Queue click for lists
# lines 7 to 3 are selected and the current line set to 7
Queue click {myListField (7,3)}

# lines 2 to 9 are selected and the current line set to 2
Queue click {myListField (2,9)}

# the current line to line 12 are selected
# the current line does not change
Queue click (Shift) {myListField (12)}
```

```
# line 13 is selected and any lines currently selected remain selected
# the current line does not change
Queue click (Shift,Command/Ctrl) {myListField (13)}

# lines 4 to 8 are selected and any lines currently selected remain selected
# the current line does not change
Queue click (Shift,Command/Ctrl) {myListField (4,8)}
```

Queue close

Command group	Flag affected	Reversible	Execute on client
Events	NO	NO	NO

Syntax

Queue close window-instance-name

Description

This command queues a “close window” event for the specified window instance as if the user had selected the close option (system menu under Windows and Linux, or close box under macOS).

The specified window instance is closed, and an evClose event is produced. If the specified window instance does not exist, the command has no effect. If you omit the window instance name, the top window instance at the time of execution will be closed, and an evClose event is generated.

Example

```
Open window instance wMyWindow/wInst1
Open window instance wMyWindow/wInst2
Queue close ## close wInst2, the top instance
```

Queue double-click

Command group	Flag affected	Reversible	Execute on client
Events	NO	NO	NO

Syntax

Queue double-click ([*Shift*][,*Command/Ctrl*]) {*field-name* (*selection-range*)}

Options

Shift	If specified, the queued event behaves as if the shift key has been pressed
Command/Ctrl	If specified, the queued event behaves as if the command/ctrl key has been pressed

Description

This command queues a “double-click event” on the specified field, that is, it simulates a user-generated double-click event on the field. A double-click event always generates an evClick before an evDoubleClick. You must specify the name of the field as a parameter, including the click positions within the field (that is, Start Row, Finish Row for lists and Start Character, Finish Character for text field selection).

There are options for including up to three modifier keys (that is, *Shift*, *Ctrl/Cmnd*) along with the click.

The field name parameter must be the name of a window field, not the name of the method associated with the field or the data name ($fieldname, not $dataname).

Queue double click returns an error if the field cannot be found. You can turn off this error by setting the option “reportQueueCommandFieldNotFoundErrors” to false, located in the “defaults” section of config.json.

Queue double-click for edit fields

Double-clicks on text within an edit field will select the complete word. If a range was specified, all COMPLETE words falling within the start and end positions will be highlighted.

Queue double-click for list fields

Double-clicks on list fields will generate an evClick followed by an evDoubleClick. The behavior in other ways is the same as described for Queue click.

Queue double-click for other field types

Pushbuttons, radio buttons, radio groups and check boxes behave in the same way as described for Queue click. An evDoubleClick event is not generated.

Example

```
# Example for edit fields
# If the text in the field is:
# Good books are the lifeblood of a master spirit
# and the command is
Queue double-click {myEditField (7,23)}
# The selected text is:
# books are the lifeblood
# Example for pushbutton - opens a new window while in Enter Data mode
# and selects all the text in the field myField
On evClick
  Open window instance wMyWindow
  Queue double-click {myField}
On default
  Quit event handler
```

Queue keyboard event

Command group	Flag affected	Reversible	Execute on client
Events	NO	NO	NO

Syntax

Queue keyboard event {*key-sequence* or *calculation*}

Description

This command queues a “keyboard” event or series of events. It simulates keyboard entry by the user from within your methods. You can enter the *key sequence* in several ways:

1. **Recording a key sequence**
   When your cursor is positioned at the end of the command, you can press the Helper button (at the bottom left of the editor panel), or press Alt-H on Windows or Cmnd+Opt+H on macOS to open the Key Sequence Recorder. You can use the **Start Recording** and **Stop Recording** buttons to specify the keys to be generated. During the recording, all key events are echoed to the Key sequence parameter field, and are not acted on by Omnis in any other way (for example, pressing Ctrl/Cmnd-Q will NOT suddenly quit Omnis). Click events, however, behave normally so you can click on **Stop recording** button.

2. **Entering into the text field**
   You can enter the text representation manually to generate the keys. Syntax checking is done at design time. When recording is off, you can edit the Key sequence parameter manually. This lets you delete key combinations or enter key sequences by hand. Since spaces are used to automatically separate key presses, the special key name SPACE will have to be manually entered to generate a "space key" event.

3. **Specifying a calculation**
   You can enter a calculation like concatenating text fields, which will contain the text representation of the keys to be generated. Syntax checking is done at runtime. Incorrect key sequence syntax will result in a runtime error. When you use a calculation, the general calculation syntax applies, which is checked at design time.

Key names

Special keys or key combinations are represented using the names of the keys. When a given key combination is run on another platform, a conversion is carried out internally so that, for example, alt-c under Windows becomes opt-c under MacOSX. The list below summarizes the conversion:

Windows and Linux

Modifier Key names: shift-, alt-, ctrl-

Special Key names: Space, Up, Down, Left, Right, PgUp, PgDn, PgLeft, PgRight, Home, End, Tab, Return, Enter, Bkspc, Clear, Cancel, Minus, Move, Del, Ins, Exit

MacOSX

Modifier Key names: shift-, opt-, com-

Special Key names: Space, Up, Down, Left, Right, PgUp, PgDn, PgLeft, PgRight, Home, End, Tab, Return, Enter, Bkspc, Clear, Cancel, Minus, Move, Del.

Set current field

If queued key events are intended for an edit field or a list, it is advisable to queue a "set current field" event before generating the key events. On the other hand, general key events, for example, menu accelerators or shortcut keys, do not require a specific current field.

Key event restrictions under Windows and Linux

Under Windows, you can use alt-<key> sequences to select menu options from the menu bar. Since the menu bar is handled by the operating environment, and **Queue keyboard event** generates internal Omnis events, queuing alt-<key> events will NOT drive the menu bar. Thus, for example, queuing alt-f will not drop the **File** menu.

As a consequence of the above restriction, evKey events are not generated for queued alt-<letter> sequences either.

A second situation where evKey events are not generated is when you queue alt-control-<letter> events. These key combinations are normally used to produce accented characters, and this facility exists only in some but not all keyboards. Since Windows does not generate character messages, these events do not generate evKey.

WARNING When queuing events on pushbuttons there is a danger of recursion under Windows and Linux, but also under macOS if buttons have been given Windows behavior, that is, they get the focus. Normally, when the focus is on a pushbutton, you can activate it by pressing the space bar. If that pushbutton receives an evClick event and has a queued space key event WITHOUT a set current field, the space key event will be sent back to the pushbutton, thereby generating another evClick, which again activates the space key event. Infinite recursion occurs, resulting in a crash.

Key event restriction under macOS

Under macOS, you use opt-<letter> to generate extended characters. When queued key events include such opt-<letter> sequences, evKey is not generated.

Example

```
# button method
On evClick
  Open window instance wMyWindow
  Queue keyboard event {y o u r n a m e}
 # paste button
On evClick
  Queue keyboard event {ctrl-V}
```

Queue OK

Command group	Flag affected	Reversible	Execute on client
Events	NO	NO	NO

Syntax

Queue OK

Description

This command queues an “OK” event. It simulates the user clicking on the OK button or pressing the Enter key.

Example

```
# trap a Tab event and issue an OK event from it
On evTab
  Queue OK
```

Queue quit

Command group	Flag affected	Reversible	Execute on client
Events	NO	NO	NO

Syntax

Queue quit

Description

This command queues a quit event. It simulates the user selecting the Exit/Quit option in the File menu. In enter data mode, a Queue OK or Queue Cancel should precede a **Queue quit** to close the enter data correctly.

Example

```
# button method to terminate data entry and quit
If flag true
  Queue OK
  Queue quit
Else
  Queue cancel
  Close top window
End If
```

Queue scroll

Command group	Flag affected	Reversible	Execute on client
Events	NO	NO	NO

Syntax

Queue scroll (*Left*|*Right*|*Up*|*Down*[,*Page*]) {*field-name* (*units*)}

Types

Left	Scroll the field to the left
Right	Scroll the field to the right
Up	Scroll the field up
Down	Scroll the field down

Options

Page	If specified, scrolling occurs in pages rather than the units corresponding to the up and down arrows on the scroll bar

Description

This command queues a "scroll" event in the specified scrollable field, that is, it simulates a mouse click or page key event on a scrollable field. With this command you can scroll a field up or down, left or right provided the appropriate scroll bar is available. You cannot use this command to scroll a window instance.

The *field name* parameter must be the name of a window field, not the name of the method associated with the field or the data name ($fieldname, not $dataname).

Queue scroll returns an error if the field cannot be found. You can turn off this error by setting the option "reportQueueCommand-FieldNotFoundErrors" to false, located in the "defaults" section of config.json.

The *Units* parameter specifies the number of lines to scroll up or down in a vertical scroll bar for a field; one unit represents one line. For a horizontal scroll bar, the unit is approximately one character.

If the *Page* option is selected, the event simulates clicking above or below the "thumb" and is the same as using the Page up or Page down key.

Example

```
# scroll a list field by 5 lines
Queue scroll (Down) {myListField (5)}
```

Queue set current field

Command group	Flag affected	Reversible	Execute on client
Events	NO	NO	NO

Syntax

Queue set current field {*field-name*}

Description

This command queues a “set current field” event in the specified field, that is, it simulates a user-generated click or tab to the specified field. In enter data mode, the contents of the field is selected. The command does not generate an evClick. However it will produce proper evBefore and evAfter events during Enter data.

The *field name* parameter must be the name of a window field, not the name of the method associated with the field or the data name ($fieldname, not $dataname).

Queue set current field returns an error if the field cannot be found. You can turn off this error by setting the option “reportQueueCommandFieldNotFoundErrors” to false, located in the “defaults” section of config.json.

Example

```
# field method to jump to another field within the same window instance
On evAfter
  Queue set current field {myField}
```

Queue tab

Command group	Flag affected	Reversible	Execute on client
Events	NO	NO	NO

Syntax

Queue tab ([*Shift*])

Options

Shift	If specified, the queued event behaves as if the shift key has been pressed

Description

This command queues a “tab” or “shift-tab” event. It simulates a user-generated tab event. With the Shift option, it simulates a shift-tab keypress.

Example

```
# Field method for a window field to simulate auto tab. When the 4th character is entered
# a tab occurs. The field must have $keyevents turned on.
On evBefore iCount as 0
On evKey
  Calculate iCount as iCount+1
  If iCount>3
    Queue tab
  End If
```

Quick check

Command group	Flag affected	Reversible	Ex
Data management	YES	NO	NO

Syntax

Quick check ([*Perform repairs*])

Options

Perform repairs	If selected, repairs to the data file are automatically carried out

Description

This command performs a quick check on the current data file. It examines the status of the current data file by reading only the internal tables in which records of any inconsistencies are stored. These records indicate corruption caused by either hardware or software failure. No attempt is made to systematically check the entire data file for problems (you use the Check data command for this purpose).

The command is not reversible: it sets the flag if it completes successfully and clears it otherwise.

If the *Perform repairs* option is specified, any repairs required are automatically carried out, otherwise the results of the check are added to the check data log. The check data log is not opened by this command but is updated if it is already open. If the Perform repairs option is specified, the following applies:

If you are not running in single user mode, Omnis automatically tests that only one user is logged onto the data file (the command fails with flag false if not), and further users are prevented from logging onto the data until the command completes.

If a working message with a count is open while the command is executing, the count will be incremented at regular intervals. The command may take a long time to execute and it is not possible to cancel execution even if a working message with cancel box is open.

Example

```
Quick check
Yes/No message {View the Quick Check log?}
If flag true
  Open check data log
End If
```

Quit all if canceled

Command group	Flag affected	Reversible	Execute on client
Methods	NO	NO	NO

Syntax

Quit all if canceled

Description

This command quits all methods that are running when the user clicks on a Cancel button inside a working message dialog box. The keyboard equivalent to the Cancel pushbutton is the Escape key under Windows and Linux, or Cmnd-period under macOS. Note that the test for cancel is carried out in Working message only if Disable cancel test at loops has first been executed.

Example

```
# Quit the current and all other methods currently running
# if the cancel button is pressed on the working message
Begin reversible block
  Disable cancel test at loops
End reversible block
Repeat
  Working message (Cancel button,Repeat count)
  Quit all if canceled
  Calculate iMyVar as iMyVar+1
Until iMyVar=500000
```

Quit all methods

Command group	Flag affected	Reversible	Execute on client
Methods	NO	NO	NO

Syntax

Quit all methods

Description

This command quits all methods that are running. If the command is executed during a method which has been called, Omnis quits both the current method and the calling method.

Example

```
# Quit all methods so that OK message never gets shown
# calling method
Do method QuitMethod
OK message {This never never gets shown}
# method Quitmethod
Quit all methods
```

Quit event handler

Command group	Flag affected	Reversible	Execute on client
Events	NO	NO	YES

Syntax

Quit event handler ([*Discard event*][,*Pass to next handler*])

Options

Discard event	If specified, Omnis discards the active event (meaning that no further processing will occur for that
Pass to next handler	If specified, Omnis will pass the event to the next level of handler (the window or task $control() me

Description

This command is used to quit out of the currently executing event handling method and is only used to terminate an On clause. It is not reversible and does not affect the flag.

If the *Discard event* option is checked, the event is thrown away and Omnis quits the event handling method.

If the *Pass to next handler* option is checked, the event is passed to the next level of handler such as the window $control() method or task $control() method.

Example

```
On evAfter
  If iName=''
    OK message {You must enter a name}
    Queue set current field {myField}
    Quit event handler (Discard event)
  End If
  # $event for a window field, to pass all events to the window $control() method
On default
  Quit event handler (Pass to next handler)
```

Quit method

Command group	Flag affected	Reversible	Execute on client
Methods	NO	NO	YES

Syntax

Quit method *return-value*

Description

This command quits the current method and returns control to the calling method, if any. If you supply a *return-value*, the command returns this value to the calling method.

Example

```
# Prompt the user to quit Omnis
Yes/No message {Do you want to quit Omnis?}
If flag true
  Quit Omnis (Force quit) ## closes all instances and tasks, then quits Omnis
End If
```

Quit Omnis

Command group	**Flag affected**	**Reversible**	**Execute on client**
Methods	NO	NO	NO

Syntax

Quit Omnis ([*Force quit*])

Options

Force quit	If specified, Omnis will force all instances to close even if they have $canclose methods that would prevent them (an

Description

This command quits Omnis closing all libraries and data files. It is equivalent to the Exit/Quit option in the File menu. However, if the Force quit option is not checked **Quit Omnis** will set the flag false and do nothing if an instance or library cannot be closed.

If the *Force quit* check box is checked Omnis will force any class instances to close so that the quit can take place, even if they have custom $canclose logic which would normally prevent them from closing.

This command can also be executed in a Web Client method running on the client. It only does anything in the Omnis Web Client running on Windows Mobile, where it quits the client application; the *Force quit* check box has no affect.

Example

Prompt the user to quit Omnis

Yes/No message {Do you want to quit Omnis?}
If flag true
  Quit Omnis (Force quit) ## closes all instances and tasks, then quits Omnis
End If

Read entire file

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

Read entire file (*path*, *binary-variable* [,*'r'*]) **Returns** *err-code*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This command reads an entire file into a binary field. If you specify 'r' as the third parameter, it opens the file in read-only mode. It returns an error code, or zero if no error occurs. The returned binary value has the following format:

- 12 byte header containing the Type (4 bytes), Creator (4 bytes), and file length (4 bytes).
- File data.

The Type is always 'TEXT', and the Creator is always 'mdos'.

Example

```
# Prompt the user for a file and read it's entire contents into the
# binary variable lBinFld
Do FileOps.$putfilename(lPathname,'Select a file','') Returns lReturnFlag
If lReturnFlag
  Read entire file (lPathname,lBinfld) Returns lErrCode
End If
```

Read file as binary

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

Read file as binary (*refnum*, *binary-variable* [,*start-position*] [,*num-bytes*]) **Returns** *err-code*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This command reads a file, or part of a file, into a binary variable. You specify the file reference number returned by the Open file command in *refnum*. The binary data read from the file is returned in *binary-variable*.

If you specify the *start-position*, the file is read at that absolute byte position (0 is the first byte in the file, 1 is the second byte in the file, and so on), otherwise it begins at the current position (0 when the file is first opened). If you specify the number of *num-bytes,* only that many bytes are read, otherwise the file is read until the end of the file is reached.

If you specify *a start-position* of 0 and *num-bytes* equal to 0, the file pointer is reset to byte position 0 in the file. If a *start-position* of -1 is given, the file pointer is reset to the end of the file. For both cases an empty binary-variable buffer is returned.

It returns an error code (See Error Codes), or zero if no error occurs. Note the special case for end of file. In this case, the command returns the error code –39, but may still have read some data.

Example

```
# Prompt the user for a file and read its contents into the binary
# variable lBinfld
Do FileOps.$putfilename(lPathname,'Select a file','') Returns lReturnFlag
If lReturnFlag
  Open file (lPathname,lRefNum)
  Read file as binary (lRefNum,lBinfld) Returns lErrCode
  Close file (lRefNum)
End If
```

Read file as character

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

Read file as character (*refnum*, *character-variable* [,*start-position*] [,*num-characters*]) **Returns** *err-code*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This command reads a file, or part of a file, into a character variable. You specify the file reference number returned by the Open file command in *refnum*. The text read from the file is returned in *character-variable*.

If you specify the *start-position*, the file is read at that absolute character position (0 is the first character in the file, 1 is the second, and so on), otherwise it begins at the current position (the first character when the file is first opened). If you specify *num-characters*, only that many characters are read, otherwise the file is read until the end of the file is reached.

If you specify a start-position of 0 and *num-characters* equal to 0, the file pointer is reset to character position 0 in the file. If a *start-position* of -1 is given, the file pointer is reset to the end of the file. For both cases an empty *character-variable* buffer is returned.

It returns an error code (See Error Codes), or zero if no error occurs. Note the special case for end of file. In this case, the command returns the error code –39, but may still have read some data.

Example

```
# Prompt the user for a text file and read its contents into the character
# variable lCharVar
Do FileOps.$putfilename(lPathname,'Select a text file','*.txt') Returns lReturnFlag
If lReturnFlag
```

```
  Open file (lPathname,lRefNum)
  Read file as character (lRefNum,lCharVar) Returns lErrCode
  Close file (lRefNum)
End If
```

ReadBinFile

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

ReadBinFile (*pathname*, *binfld* [, *start* [, *length*]]) **Returns** *return-value*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

ReadBinFile reads binary data from the file system or data fork (not the resource fork).

Note for macOS Users: ReadBinFile and WriteBinFile are only useful for reading and writing the data fork of files.

Pathname is a character field containing the full path of the file to read.

Binfld is a binary field in which the data is stored.

Start is an optional parameter specifying an integer field that contains the byte position in the file where the command should start reading. Defaults to 0 (zero), that is, the beginning of the file.

Length is an optional parameter specifying an integer field containing the number of bytes to read. If the parameter is not used, the value defaults to the length of the file.

Return-value is a long Integer that is the number of bytes read, if no error occurs. Otherwise, it is an error code, one of:

Code
-1
-2
-10
-11
-12
-20
-100
-101
-998

Example

```
# read the binary data from the file lPathname
Calculate lPathname as con(sys(115),'binfile')
ReadBinFile (lPathname,lBinfld) Returns lNumbytes
OK message {[lNumbytes] bytes read}
```

Redefine list

Command group	Flag affected	Reversible	Execute on client
Lists	NO	NO	NO

Syntax

Redefine list {*list-of-field-or-file-names* (F1,F2..F3,F4)}

Deprecated Command

This command has been deprecated and is no longer visible in the Code Assistant in the Code Editor (it will not appear when you type the first few characters), although it is still present in Omnis Studio and will continue to function if used in legacy code. You can show this command by disabling the appropriate Command Filter in the **Modify** menu in the Code Editor.

Description

This command redefines the columns of the current list. No change is made to the data currently stored in the list, or to the data type of each column; the command only changes the field name associated with each column. If you pass more field names to **Redefine list** than the current number of columns, the extra names are ignored. List boxes on windows will no longer display the data in the list unless you change their $calculation property to include the new variable or field name(s).

Example

```
Set current list iList1
Define list {iCol1Date,iCol2Num,iCol3Char}
Add line to list
# redefine the 3rd column in the list to hold boolean values
Redefine list {iCol1Date,iCol2Num,iCol4Boolean}
# the boolean field value is converted into a character field format 'YES' etc and then add to the list
Add line to list
# or do it like this
Do ilist.$redefine(iCol1Date,iCol2Num,iCol4Boolean)
```

Redraw

Command group	Flag affected	Reversible	Execute on client
Fields	YES	NO	NO

Syntax

Redraw ([Refresh now]) {list-of-field-or-window-names (Name1,Name2,...)}

Options

Refresh now	If specified, the redraw occurs before the command finishes executing, rather than occurring at a later indeterminate point

Description

This command redraws the specified field or window instance (or list of fields or window instances). The *Refresh now* option ensures the redraw is completed when the command is executed. Without this option the redraw occurs when the method has finished executing.

Example

```
Prepare for edit
Enter data
If flag true
  Update files
Else
  Clear main & con
  Redraw {wDataEntry}
End If
# alternatively you can use the $redraw(setcontents,refresh) method to redraw the
# contents and/or refresh a field or window# setcontents defaults to true, refresh to false
Do $cfield.$redraw() ## redraw current field
Do $cwind.$redraw() ## redraw current window
Do $root.$redraw() ## redraw all window instances
```

Redraw lists

Command group	Flag affected	Reversible	Execute on client
Fields	NO	NO	NO

Syntax

Redraw lists ([*All windows*][,*All lists*][,*Selection only*])

Options

All windows	If specified, the command applies to all open window instances, rather than just the top open window instance
All lists	If specified, the command applies to all lists on the window instance(s),rather than just the current list
Selection only	If specified, the command redraws the lists without reloading the data, in order to show changes to the selection state

Description

This command redraws the current list window field or all list fields. It lets you update the display of the current list field after you delete, change, or insert a line, so that the screen list reflects the changes. When Omnis executes **Redraw lists**, the selected line is scrolled into view and the visible lines recalculated.

Omnis can execute a **Redraw lists** command for all window instances and for all lists using the *All windows*, and *All lists* options. If neither option is selected, only the fields on the top window instance which display the current list are redrawn.

The *Selection only* option causes the redraw to affect the highlighting of the selected lines, the contents are not redrawn.

Omnis also redraws any fields which are local to the list field so that they will display the new values. It also redraws the grid fields associated with the current list.

Example

```
Begin reversible block
  Set current list iList
End reversible block
Define list {iCol1,iCol2}
Calculate iCol1 as 42
Add line to list {(iCol1,chr(iCol1))}
Redraw lists
```

Redraw menus

Command group	Flag affected	Reversible	Execute on client
Menus	NO	NO	NO

Syntax

Redraw menus

Description

This command redraws all instances of your own custom menus. When executing **Redraw menus**, Omnis re-evaluates any square-bracket notation contained in the menu titles and lines before redrawing the menu bar.

Example

```
# Redraw the menus so that any with the title set to
# [tMenuName] are updated with the new menu name
Calculate tMenuName as 'MyMenu'
Redraw menus
```

Redraw toolbar

Command group	Flag affected	Reversible	Execute on client
Toolbars	NO	NO	NO

Syntax

Redraw toolbar ([*Droplists only*]) {*toolbar-instance*}

Options

Droplists only	If specified, the command will only redraw the droplists on the toolbar

Description

This command redraws the toolbar instance. You can redraw droplists only using the *Droplists only* option.

Example

```
Show docking area {kDockingAreaLeft}
Install toolbar {tbMyToolbar}
# do something
# then redraw the droplists displayed on the toolbar tbMyToolbar
Redraw toolbar (Droplists only) {tbMyToolbar}
```

Redraw working message

Command group	Flag affected	Reversible	Execute c
Message boxes	NO	NO	NO

Syntax

Redraw working message

Description

This command redraws the text in the working message after evaluating any square bracket notation. Omnis does not increment the working message count and does nothing if there is no open working message.

Example

```
# Redraw the working message to update the record counter
Working message {Processing Record [lCount]}
For lCount from 1 to 20000 step 1
  Redraw working message
End For
```

Register DLL

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

Register DLL (*library*, *procedure*, *type-definition* [,*unregister* {Default kFalse}]) **Returns** *err-code*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This command registers a procedure in a DLL, so that you can use Call DLL to call it.

The *library* is the name or pathname of the DLL containing the procedure specified by procedure. If you do not specify a pathname for the library then the standard operating system search rules for DLLs will apply.

The *type-definition* specifies the data types of the return value and parameters required by procedure. t*ype-definition* is a character string:

The first character identifies the data type of the return value of the procedure

The remaining characters (one for each parameter) identify the parameter data types

Register and Call DLL commands support 64-bit type specifiers. The following table lists the possible data type characters:

Character	Description	Pass By	C declaration
A	Logical	Value	short int
B	IEEE 8 byte floating point	Value	double
C	Null-terminated string	Value	TCHAR *
D	Pascal string	Value	TCHAR *
E	IEEE 8 byte floating point	Reference	double *
H	Unsigned 16 bit integer	Value	unsigned short int
I	Signed 16 bit integer	Value	short int

Character	Description	Pass By	C declaration
J	Signed 32 bit integer	Value	long int
L	Logical	Reference	short int *
M	Signed 16 bit integer	Reference	short int *
N	Signed 32 bit integer	Reference	long int *
O	Null-terminated 8-bit string: Encoded as Ansi on Win32, MacRoman on macOS, ISO-8859-1 on Linux	Value	char *
P	Signed integer of pointer size for the current architecture (32 or 64 bit)	value	Intptr_t
Q	Signed integer of pointer size for the current architecture (32 or 64 bit)	Reference	Intptr_t *
R	Signed 64-bit integer	Value	Int64_t
S	Signed 64-bit integer	Reference	Int64_t *
V	void (use if no return value)		void
Z	Cannot be used as a return value data type. When passing the parameter, behaves like data type C. Use Z to indicate that the procedure sets the parameter to a sequence of null-terminated strings,terminated by an additional null character. *Call DLL* sets the parameter to the same sequence of strings,using carriage returns instead of null terminators	Value	char *

Note that TCHAR represents the character type used for operating system API calls - for Windows it is 16 bit unsigned short.

When you have finished using the procedure, you may wish to unregister the procedure; this allows Omnis to unload the DLL containing the procedure. To do this, pass the unregister parameter as kTrue. Note that the DLL will remain loaded until all the registered procedures in the DLL have been unregistered; also, if the DLL is loaded into the Omnis process for another reason e.g. the DLL is linked with the Omnis executable, then it will remain loaded even after the last procedure has been unregistered.

Example

```
# Flash the Omnis window to attract the user's attention
# Win32 API to get the main Omnis window: HWND GetActiveWindow(VOID)
Register DLL ('USER32.DLL','GetActiveWindow','J')
Call DLL ('USER32.DLL','GetActiveWindow') Returns lHWND
# Win32 API to Flash a window: BOOL FlashWindow(HWND, BOOL)
Register DLL ('USER32.DLL','FlashWindow','JJJ')
Call DLL ('USER32.DLL','FlashWindow',lHWND,1) Returns lResult
```

This example creates a file and loads the contents:

```
Register DLL ("KERNEL32.DLL","CreateFileA","JCJJJJJJ")
Register DLL ("KERNEL32.DLL","CloseHandle","JJ")
Register DLL ("KERNEL32.DLL","ReadFile","J,J,C32768,J,N,J")
Call DLL ("KERNEL32.DLL","CreateFileA","c:\MYBIGFILE.TXT",-1073741824,3,0,3,268435584,0) Returns #1
Call DLL ("KERNEL32.DLL","ReadFile",#1,#S1,32767,#49,0) Returns #50
Call DLL ("KERNEL32.DLL","CloseHandle",#1) Returns #50
Calculate #1 as binlength(#S1)
```

Reinitialize search class

Command group	Flag affected	Reversible	Execute on client
Searches	NO	NO	NO

Syntax

Reinitialize search class

Description

This command reloads the current search definition into memory. **Reinitialize search class** is useful if square bracket notation has been used in the search class. The square bracket expressions are re-evaluated using current field values before reloading the search definition. Each find table keeps its own copy of the search conditions so you must re-issue the Find command if a search needs reinitializing.

Example

```
# This example assumes a search class sTown uses the comparison line TOWN Begins with [lStartsWith]
# The window wStarts is used to allow the user to specify a value for #S5
Set search name sTown
Repeat
  Open window instance wStarts/CEN
  Enter data
  Close window instance wStarts
  If flag true
    Reinitialize search class
    Do method PrintReport
  End If
Until flag false
```

Remove all menus

Command group	Flag affected	Reversible	Execute on client
Menus	NO	YES	NO

Syntax

Remove all menus

Description

This command removes all menu instances from the menu bar, excluding the standard Omnis menus such as File and Edit. If you use **Remove all menus** in a reversible block, the menu instances are reinstalled when the method containing the block finishes.

Example

```
# Remove all user defined menus from the main
# omnis menubar
Begin reversible block
  Remove all menus
End reversible block
OK message {Menus are now removed}

# now all menu instances are reinstalled
```

Remove final menu

Command group	Flag affected	Reversible	Execute on client
Menus	NO	YES	NO

Syntax

Remove final menu

Description

This command removes the final or right-most menu instance from the menu bar, excluding the standard Omnis menus such as File and Edit. If you use **Remove final menu** in a reversible block, the final menu instance is reinstalled when the method containing the block terminates.

Example

```
# Remove the last menu installed
Begin reversible block
  Remove final menu
End reversible block
OK message {Menu is now removed}
# now the final menu is reinstalled
```

Remove menu

Command group	Flag affected	Reversible	Execute on client
Menus	YES	YES	NO

Syntax

Remove menu menu-instance-name

Description

This command removes the specified menu instance from the menu bar and sets the flag. You can choose the menu name from a list containing any custom and standard built-in menus, such as *File, *Edit, and so on.

If you use this command to remove a menu instance which has previously been installed in place of the standard File or Edit menu (using the Replace standard File menu or Replace standard Edit menu command) the previously replaced standard File or Edit menu is restored.

If you use **Remove menu** in a reversible block, the specified menu instance is reinstalled when the method containing the reversible block terminates.

Example

```
# If the menu mView is installed remove it
Test for menu installed {mView}
If flag true
  Remove menu mView
End If
# Alternatively, you can remove a menu using $close
Do $imenus.mView.$close()
```

Remove toolbar

Command group	Flag affected	Reversible	Execute on client
Toolbars	NO	NO	NO

Syntax

Remove toolbar {*toolbar-instance*}

Description

This command removes the specified toolbar instance.

Example

```
Show docking area {kDockingAreaLeft}
Install toolbar {tbMyToolbar/kDockingAreaLeft}
# do something
# now remove the toolbar & docking area again
Remove toolbar {tbMyToolbar}
Hide docking area {kDockingAreaLeft}
# or you do it like this
Do $itoolbars.tbMyToolbar.$close()
# or if you used notation to install the toolbar
Do $clib.$toolbars.tbMyToolbar.$open('*',kDockingAreaLeft) Returns lToolBarRef
# use the toolbar reference to close it
Do lToolBarRef.$close()
```

Rename class

Command group	Flag affected	Reversible	Execute on client
Classes	YES	NO	NO

Syntax

Rename class ([Perform find and replace]) {class-name/new-name}

Options

Perform find and replace	If specified, and the command is executing in a development version of Omnis, the command opens the find and replace dialog to allow the user to replace the old class name with the new class name

Description

This command renames the specified library class and can perform a find and replace. Errors, such as attempting to use a name that is already in use, simply clear the flag and display an error message. You can rename a class which is in use.

When renaming a class, you can use the Perform find and replace option to search through all the classes in the library and replace the references to the old class name with the new name.

Example

```
New class {Search Class/sMySearch} ## create new search class
Modify class {sMySearch} ## let user modify it
Delete class {sUser} ## delete the search class sUser
Rename class {sMySearch/sUser} ## rename the new search class to the old search
Set search name sUser
Print report (Use search)
```

Rename data

Command group	Flag affected	Reversible	Ex
Data management	YES	NO	NO

Syntax

Rename data {*file-name/new-slot-name*}

Description

This command renames the data for a specified file class in a data file so that the data will then belong to a file with a different name; that is, it renames a slot. The existing file class name and the new slot name are specified as parameters.

The specified file class is disconnected from the data, and an empty slot and indexes for that file will be created as soon as that file is accessed again.

If the specified file name does not include a data file name as part of the notation, the default data file for that file is assumed.

If the file is closed or memory-only, the command does not execute and returns flag false.

If you are not running in single user mode, Omnis automatically tests that only one user is logged onto the data file (the command fails with flag false if this is not true), and further users are prevented from logging onto the data until the command completes.

This command sets the flag if it completes successfully and clears the flag otherwise. The command is not reversible.

Example

```
Rename data {fCustomers/fCustomersArchive}
If flag true
 OK message {File archived}
Else
 OK message {Cannot archive while more than 1 user is logged on}
End If
```

Reorganize data

Command group	Flag affected	Reversible	Ex
Data management	YES	NO	NO

Syntax

Reorganize data ([*Test only*][,*Optimize*][,*Convert pictures*][,*Use true color when converting*]) {*list-of-files* (*F1,F2,..,Fn*) (leave empty to select all)}

Options

Test only	If specified,the data file is not updated; the command purely tests to see if it would update the data file when executed without this option specified,and returns the flag set to true if an update would occur
Optimize	If specified,the command also attempts to optimize free space to make data storage more efficient
Convert pictures	If specified,the command also converts (where necessary) pictures to shared (cross-platform) picture format
Use true color when converting	Only relevant when 'Convert pictures' is specified.If specified,pictures are converted to the recommended shared true color format,rather than the out of date shared 256 color format.

Description

This command reorganizes the data for the specified file or list of files. Reorganization is the process by which the data structures held in the Omnis data file are brought into line with the file class definitions.

Reorganize data reorganizes the data for the specified list of files, and is equivalent to the option on the Slot menu in the Data File Browser.

If you omit a file name or list of files, all the files with slots in the current data file are reorganized.

If a specified file name does not include a data file name as part of the notation, the default data file for that file is assumed. If the file is closed or memory-only, the command does not execute and returns with the flag false.

If you are not running in single user mode, Omnis automatically tests that only one user is logged onto the data file (the command fails with the flag false if this is not true), and further users are prevented from logging onto the data until the command completes.

If a working message with a count is open while the command is executing, the count will be incremented at regular intervals. The command may take a long time to execute, and it is not possible to cancel execution even if a working message with cancel box is open.

The command sets the flag if it completes successfully and clears the flag otherwise. The command is not reversible.

If the *Test only* checkbox option is specified, no reorganization is actually carried out. The flag is set if at least one file needs reorganization.

The *Optimize* checkbox option specifies whether reorganize with optimize is to be carried out. This distributes the free space to make the data storage more efficient.

The *Convert pictures* checkbox option causes all pictures in the data to be converted to a shared picture format.

Example

```
Reorganize data (Test only) ## all files
If flag true
```

```
  Yes/No message {Reorganize now?}
  If flag true
    Reorganize data
  End If
Else
  OK message {No reorganization required}
End If
```

Repeat

Command group	Flag affected	Reversible	Execute on client
Constructs	NO	NO	YES

Syntax

Repeat

Description

This command repeats a command or series of commands that are contained in a loop closed by an Until command. Each time the command is repeated, Omnis tests the condition attached to the Until command to ensure that the condition is true. If the condition is true, the commands in the loop are not executed and the command after the Until is executed. However, if the condition is false, Omnis jumps back to the first command following the **Repeat** command.

An error will result if there is a **Repeat** command without a matching Until command. **Repeat** loops always execute at least once. The Repeat– Until logic test is carried out at the end of the loop, after the commands in the loop are executed, whereas the While–End While logic test is carried out at the beginning of the loop.

Example

```
Repeat
  Yes/No message {Press Yes to exit loop}
Until flag true
Repeat
  No/Yes message {Press No to exit loop}
Until flag false
Repeat
  Prompt for input Enter a value greater than 10 to exit loop Returns lValue
Until lValue>10
```

Replace line in list

Command group	Flag affected	Reversible	Execute on client
List lines	YES	NO	NO

Syntax

Replace line in list {*line-number* (*values*) {default is current line}}

Deprecated Command

This command has been deprecated and is no longer visible in the Code Assistant in the Code Editor (it will not appear when you type the first few characters), although it is still present in Omnis Studio and will continue to function if used in legacy code. You can show this command by disabling the appropriate Command Filter in the **Modify** menu in the Code Editor.

Description

This command transfers field values from the current record buffer to the corresponding fields in the current list. Alternatively, it is possible to specify a comma-separated list of values enclosed in brackets after the line number. In this case, the values stored in the specified line of the list are set up from the values in the brackets and not from the variables specified when the list was defined.

Replace line in list {LIST.$linecount('abc',,LVAR12+3)}

will store 'abc' into the first column of the final line of the current list, leave the value of the second column unchanged, and load the result of LVAR12+3 into the third column. If too few values are specified, the other columns will be left unchanged; if too many values are specified, the extra values are ignored. Any conversions required between data types are carried out.

If the line number specified in the command line is empty, or if it evaluates to zero, the current line is used. If the list is empty or if the line is beyond the current end of the list, the flag is cleared.

Example

```
# Replace Harry with Arnold and increment
# the age of everybody in the list by 1
Set current list lMyList
Define list {lName,lAge}
Add line to list {('Fred',10)}
Add line to list {('George',20)}
Add line to list {('Harry',22)}
Add line to list {('William',31)}
Add line to list {('David',62)}
Replace line in list {3 ('Arnold',47)}
For each line in list from 1 to lMyList.$linecount step 1
  Load from list
  Calculate lAge as lAge+1
  Replace line in list {(,lAge)}
End For
```

Replace standard Edit menu

Command group	Flag affected	Reversible	Execute on client
Menus	NO	YES	NO

Syntax

Replace standard Edit menu {*class-name*[/*instance-name*] [(*parameters*)]}

Description

This command removes the standard built-in Edit menu from the menu bar and replaces it with a custom menu. You can assign an instance name for the replacement menu. The default instance name of the replacement menu is the menu class name. If no replacement menu name is specified, the Edit menu is reinstated.

The replacement menu will remain enabled even when commands such as Disable all menus are issued, or modal user-defined windows are opened. The only time the replacement menu will not remain enabled is when a report is printed to screen with Send to screen, and the check box option Do not wait for user is not checked (that is, Omnis is awaiting user input).

You can disable the Edit menu or its replacement menu by using Disable menu line.

Example

```
# Replace the standard edit menu with the user
# defined menu mMyEdit while in enter data
# $construct of window
```

```
Replace standard Edit menu {mMyEdit}
Enter data
Replace standard Edit menu ## put system Edit menu back
```

Replace standard File menu

Command group	Flag affected	Reversible	Execute on client
Menus	NO	YES	NO

Syntax

Replace standard File menu {*class-name*[/*instance-name*] [(*parameters*)]}

Description

This command removes the standard built-in File menu from the menu bar and replaces it with a custom menu. You can assign an instance name for the replacement menu. The default instance name of the replacement menu is the menu class name. If no replacement menu name is specified, the File menu is reinstated.

The replacement menu will remain enabled even when commands such as Disable all menus are issued, or modal user-defined windows are opened. The only time the replacement menu will not remain enabled is when a report is printed to screen with the Send to screen command, and the check box option *Do not wait for user* is not checked (that is, Omnis is awaiting user input).

You can disable the File menu or its replacement menu by using Disable menu line.

Example

```
# Replace the standard file menu with the user
# defined menu mMyFile while in enter data
# $construct of window
Replace standard File menu {mMyFile}
Enter data
Replace standard File menu ## put system File menu back
```

Request advises

Command group	Flag affected	Reversible	Ex
Exchanging data	YES	YES	NO

Syntax

Request advises *field-name* {*server-data-item-name*}

Description

DDE command, Omnis as client. This command sends a request to the server asking to be advised of any changes made to a specified data item. An error occurs if the channel is not open. The command takes the Omnis field name and the server data item name as parameters. The data item name can contain square bracket notation.

Whenever Omnis is advised of a change in field value, that value is changed providing your library is in enter data mode.

The flag is set if the command is successful.

You can use a control method to detect the arrival of data from the server using evSent.

Example

```
Request advises iCompany {iCompany}
Request advises iAddress {iAddress}
Prepare for insert
Enter data
Update files if flag set
```

Request field

Command group	Flag affected	Reversible	Execute
Exchanging data	YES	NO	NO

Syntax

Request field *field-name* {*server-data-item-name*}

Description

DDE command, Omnis as client. This command requests a data item from the DDE channel. An error occurs if the channel is not open. The command takes the Omnis field name and the server data item name as parameters. The data item name can contain square bracket notation. If the data item name is not specified, the Omnis field name is used. The flag is set if the command is successful.

Example

```
Set DDE channel number {1}
Calculate lAttempts as 1
# keeps trying until conversation opened or number of attempts > 10
Repeat
  Open DDE channel {Omnis|DDE2}
  Calculate lAttempts as lAttempts+1
Until #F|lAttempts>10
Calculate iCommand as '[TakeControl]'
Send command {[iCommand]}
If flag false
  OK message {Error: [iCommand], Open Attempts = [lAttempts]}
End If
Request field iCompany {iCompany}
Request field iAddress {iAddress}
Prepare for insert with current values
Enter data
Update files if flag set
```

Restore selection for line(s)

Command group	Flag affected	Reversible	Execute on client
List lines	YES	NO	NO

Syntax

Restore selection for line(s) ([*All lines*]) {*line-number* (*calculation*)}

Options

All lines	If specified, the command affects all the lines in the list

Deprecated Command

This command has been deprecated and is no longer visible in the Code Assistant in the Code Editor (it will not appear when you type the first few characters), although it is still present in Omnis Studio and will continue to function if used in legacy code. You can show this command by disabling the appropriate Command Filter in the **Modify** menu in the Code Editor.

Description

This command copies the Saved selection state to the Current selection state and sets the flag. To allow sophisticated manipulation of data via lists, a list can store two selection states for each line; the "Current" and the "Saved" selection. The Current and Saved selections have nothing to do with saving data on the disk; they are no more than labels for two sets of selections. The lists may be held in memory and never saved to disk: they will still have a Current and Saved selection state for each line but they will be lost if not saved. When a list is stored in the data file, both sets of selections are stored.

The **Restore selection for line(s)** command allows the Saved selection state of the specified line (or All lines) to be copied into the Current set. You can specify a particular line in the list either by entering a number or a calculation. You are required to redraw the list to refresh the state of the displayed list field. The *All lines* option restores the selection states for all lines of the current list.

Example

```
# Save and restore the selection after all
# lines have been deselected
Set current list lMyList
Define list {lCol1}
For lCol1 from 1 to 6 step 1
  Add line to list {lCol1}
End For
Select list line(s) {3}
Select list line(s) {5}
Save selection for line(s) (All lines)
Deselect list line(s) (All lines)
Restore selection for line(s) (All lines)
```

Revert class

Command group	Flag affected	Reversible	Execute on client	P
Classes	YES	NO	NO	A

Syntax

Revert class {*class-name*}

Description

This command reads the specified class from the library file on disk into RAM, so that any changes made to that class using the notation are lost. The flag is set if the class is successfully re-read. A runtime error occurs if the specified class cannot be found.

Example

```
# make change to a window class
Do $windows.wMyWindow.$objs.Field1.$visible.$assign(kFalse)
Open window instance wMyWindow
# do something
Revert class {wMyWindow} ## reset the Field1 on the saved window (NOT the current instance) to be visible
```

Save class

Command group	Flag affected	Reversible	Execute on client
Classes	YES	NO	NO

Syntax

Save class {*class-name*}

Description

This command writes the specified class, which normally contains changes made by notation, into the library file on disk. You use **Save class** to make the changes permanent. The flag is set if the class is successfully saved. A runtime error occurs if the specified class cannot be found.

Example

```
# get a reference to a window in the current library
Set reference lWinRef to $windows.wMyWindow
# create a pushbutton object on the window
Set reference lObjRef to lWinRef.$objs.$add(kPushbutton,5,5,23,120)
# save the class
Save class {wMyWindow}
# opens the window with the new button
Open window instance wMyWindow
```

Save selection for line(s)

Command group	Flag affected	Reversible	Execute on client
List lines	YES	NO	NO

Syntax

Save selection for line(s) ([*All lines*]) {*line-number* (*calculation*)}

Options

All lines	If specified, the command affects all the lines in the list

Deprecated Command

This command has been deprecated and is no longer visible in the Code Assistant in the Code Editor (it will not appear when you type the first few characters), although it is still present in Omnis Studio and will continue to function if used in legacy code. You can show this command by disabling the appropriate Command Filter in the **Modify** menu in the Code Editor.

Description

This command saves the selection state of the specified line(s) in memory and sets the flag. To allow sophisticated manipulation of data via lists, a list can store two selection states for each line; the “Current” and the “Saved” selection. The Current and Saved selections have nothing to do with saving data on the disk; they are no more than labels for two sets of selections. The lists may be held in memory and never saved to disk: they will still have a Current and Saved selection state for each line but they will be lost if not saved. When a list is stored in the data file, both sets of selections are stored.

Save selection for line(s) allows the selection state of the specified line (or All lines) to be copied into the Saved set. You can specify a particular line in the list by entering either a number or a calculation. If the line number is not specified, the current line selection is saved. The *All lines* option saves the selection for all lines of the current list. This example selects the middle line of the list:

Example

```
# Save and restore the selection after all an invert
Set current list lMyList
Define list {lCol1}
For lCol1 from 1 to 6 step 1
  Add line to list {lCol1}
End For
Select list line(s) {3}
Select list line(s) {5}
Save selection for line(s) (All lines)
Invert selection for line(s) (All lines)
Restore selection for line(s) (All lines)
```

SEA continue execution

Command group	Flag affected	Reversible	Execut
Error handlers	NO	NO	NO

Syntax

SEA continue execution

Description

This command continues method execution at the command following the command which called an error handler; SEA stands for Set Error Action. Using it is, in effect, like saying “Error is acknowledged. Now, skip over the error line and proceed with the succeeding good lines.”

Using this command is similar to setting the go point in the debugger at L+1 where L is the error line. The command is always used within an error handler.

Example

```
# error handler to trap break key while waiting for semaphore
# The edit method must test the flag to prevent an error on update
If #ERRCODE=kerrCantlock
  OK message {User cancelled request for record lock}
  SEA con execution
End If
```

SEA repeat command

Command group	Flag affected	Reversible	Execut
Error handlers	NO	NO	NO

Syntax

SEA repeat command

Description

This command attempts to repeat the command that caused an error; SEA stands for Set Error Action. This is most useful after an out of memory condition. The command is always used within an error handler. It is your responsibility to ensure that an endless looping situation between the error handler and the command is not created. Also, you must ensure that any side effects of the original execution of the command which caused the error are taken into account.

Example

```
# error handler traps an attempt to edit a locked record and the user presses the Break key
If #ERRCODE=kerrCantlock
  Yes/No message {Do you want to cancel the edit?}
  If flag true
    Quit all methods
  Else
    SEA repeat command
  End If
End If
```

SEA report fatal error

Command group	Flag affected	Reversible	Execut
Error handlers	NO	NO	NO

Syntax

SEA report fatal error

Description

This command causes the default action for a fatal error to occur; SEA stands for Set Error Action. If the debugger is available, it is invoked, otherwise, execution halts with an error message. This command, like the other SEA commands, should only be used from within an error handler. The SEA commands determine the behavior following fatal or warning errors.

Example

```
# This causes a warning error to generate the same action as a fatal error
If #ERRCODE=kerrUnqindex
  SEA report fatal error
  # your code...
End If
```

Search list

Command group	Flag affected	Reversible	Execute on client
Lists	YES	NO	NO

Syntax

Search list ([*From start*][,*Only test selected lines*][,*Select matches (OR)*][,*Deselect non-matches (AND)*][,*Do not load line*])

Options

From start	If specified, the command starts with the first line of the list rather than the line immediately after the current line
Only test selected lines	If specified, the command only operates on selected lines

Select matches (OR)	If specified, the command processes all specified lines and selects lines which match the search; any lines selected before the command executes remain selected
Deselect non-matches (AND)	If specified, the command processes all specified lines and deselects lines which do not match the search
Do not load line	If specified, the line found by the search is not loaded into the current record buffer; this is only relevant when 'Select matches (OR)' and 'Deselect non-matches (AND)' are both not specified

Deprecated Command

This command has been deprecated and is no longer visible in the Code Assistant in the Code Editor (it will not appear when you type the first few characters), although it is still present in Omnis Studio and will continue to function if used in legacy code. You can show this command by disabling the appropriate Command Filter in the **Modify** menu in the Code Editor.

Description

This command searches the current list for field values that match the current search class or search calculation and loads them into the Current Record Buffer. The search starts at the beginning of the list if *From start* is checked, otherwise at the line after the current line.

If Omnis finds a line that matches the search class, that line number becomes the current line $line and the flag is set. If Omnis cannot find a matching line, the $line is cleared and the flag is cleared. If there is no current search class, all lines are said to match and Omnis sets the flag.

When checked, the *Do Not Load Line* option ensures the line found by the search is not loaded into the current record buffer.

The *Only test selected lines* option restricts the list scan to selected lines only. If the *Select matches (OR)* option is checked, the command scans all the lines from the line after the current line to the end and selects all those that match the search; if you also use the *From start* option, the whole of the list is scanned, that is, the search starts at line 1. Lines that are already selected before the command is executed remain selected. This is equivalent to ORing the existing selected lines with the lines that match the search. The current line is not affected.

If the *Deselect non-matches (AND)* option is used, the command scans all the lines from the line after the current line to the end and deselects all those which do not match the search; if you also use the *From start* option, the whole of the list is scanned, that is, the search starts at line 1. Lines which are already selected before the command is executed are deselected if they do not match the search, that is, the only lines left selected are those which were already selected and which match the search. This is equivalent to ANDing the existing selected lines with the lines which match the search. The current line is not affected.

Using the Select and the Deselect options together alters the selection state so that matching lines are selected, non-matching lines are deselected. The current line is not affected.

Example

```
Set current list iList1
Define list {iColNum}
Calculate iColNum as 1
Repeat
  Add line to list
  Calculate iColNum as iColNum+1
Until iColNum=6
Set search as calculation {iColNum=3|iColNum=4}
Search list (From start) ## current line is now 3
```

```
Search list (Select matches (OR)) ## selects line 4
# or do it like this
Do iList1.$search(iColNum=3|iColNum=4,kTrue,kFalse,kTrue,kFalse)
Do iList1.$first(kTrue)
```

Select list line(s)

Command group	Flag affected	Reversible	Execute on client
List lines	YES	NO	NO

Syntax

Select list line(s) ([*All lines*]) {line-number (calculation)}

Options

All lines	If specified, the command affects all the lines in the list

Deprecated Command

This command has been deprecated and is no longer visible in the Code Assistant in the Code Editor (it will not appear when you type the first few characters), although it is still present in Omnis Studio and will continue to function if used in legacy code. You can show this command by disabling the appropriate Command Filter in the **Modify** menu in the Code Editor.

Description

This command selects the specified list line. The specified line of the current list is selected and is shown highlighted (or checked on popup lists) on any window list fields provided that the field has $multipleselect on. If the line number is not specified, the current list line is selected. The *All lines* option selects all lines of the current list. The current line is not affected. When a list is saved in the data file, the line selection is stored. The following example selects the middle line of the list:

Example

```
# Select line 3 of the list
Set current list lMyList
Define list {lCol1}
For lCol1 from 1 to 6 step 1
  Add line to list {lCol1}
End For
Select list line(s) {lMyList.$linecount/2}
# Alternatively, you can select a line by assigning its $selected property.
Do lMyList.1.$selected.$assign(kTrue)
```

Select printer

Command group	Flag affected	Reversible	Execute on client
Reports and Printing	YES	NO	NO

Syntax

Select printer ([*Discard previous settings*]) {printer-name}

Options

Discard previous settings	If specified, the command reloads the Omnis page setup with the default system settings for the selected printer (Windows platform only)

Description

This command allows the user to specify a printer to receive reports. You can choose the required printer from a list of all installed printers. After this command has executed, the flag is set if the printer was selected successfully.

The *Discard previous settings* option causes Omnis to reload the Omnis page setup with the default system settings for the specified printer.

You can use the function *sys(101)* to return the name of the current printer.

Example

```
# Select the printer prior to printing
Select printer {MyPrinter}
If flag true
  Set report name rMyReport
  If flag true
    Print report
  End If
End If
```

Send advises now

Command group	Flag affected	Reversible	Ex
Exchanging data	NO	NO	NO

Syntax

Send advises now

Description

DDE command, Omnis as server. This command advises the client applications of all the field values for all the fields for which Advise requests have been received. The values are taken from the CRB.

Example

```
Set main file {fCustomers}
Find on fCustomers.CustomerID (Exact match) {iCustID}
Send advises now
```

Send command

Command group	Flag affected	Reversible	Ex
Exchanging data	YES	NO	NO

Syntax

Send command {*text*}

Description

DDE command, Omnis as client. This command sends a command or a series of commands as text to the current channel.

The command-text syntax must conform to whatever syntax rules apply to the server program.

The DDE syntax dictates that the commands be enclosed in square brackets and Omnis attaches special meaning to them in strings. Therefore, it may be necessary to put the command text into one of the Omnis string variables.

The flag is set if the server accepts the command(s).

Syntax and errors

When you send commands to Omnis, the syntax is defined by the text shown in the method editor. You can enter scripts in Omnis, copy them to the clipboard and paste them into the client application. If the sent command returns an error to Omnis, the hash variables *#ERRCODE* and *#ERRTEXT* store the error code and message.

Example

```
# put your command text into the character variable lString
Calculate lString as '[your command]'
Send command {[lString]} ## send the command to the server
# else you can enter the command directly into the command parameter by doubling the first set of brackets# eg
Send command {[[releasecontrol]}
# Example
Set DDE channel number {2}
Open DDE channel {Omnis|Country}
If flag false
  OK message {The Country library is not running}
Else
  Calculate lString as "Ok Message {Hi, this is DDE magic'}"
  Send command {[lString]}
  Send command {'Next'}
  Close DDE channel
  OK message {Update finished}
End If
```

Send field

Command group	Flag affected	Reversible	Ex
Exchanging data	YES	NO	NO

Syntax

Send field field-name {server-data-item-name}

Description

DDE command, Omnis as client. This command sends the value of an Omnis field to the current DDE channel. An error occurs if the channel is not open. The command takes the Omnis field name and the server data item name as parameters. The data item name can contain square bracket notation. If the data item name is not specified, the Omnis field name is used.

The flag is set if the server program accepts the value.

Example

```
Set DDE channel number {2}
Open DDE channel {Omnis|DDE2}
Calculate lString as '[TakeControl]'
Send command {[lString]}
If flag false
  OK message {Error sending: [lString]}
End If
Send field iClient {sName}
Send field iTotal {sTotals}
```

Send to a window field

Command group	Flag affected	Reversible
Report destinations	NO	YES

Syntax

Send to a window field {*field-name*}

Description

This command directs the output of a report to a window Screen Report field; you cannot print to any other type of window field. When you print the report the field is changed into a standard screen report window that has all the features of the standard screen report.

An error is generated if the field name is invalid for the current window. If you use **Send to a window field** in a reversible block, the report destination reverts to its former setting when the method terminates.

Example

```
# the $event method behind a pushbutton on a window
On evClick
  Send to a window field {WindowReportField}
  Set report name rMyReport
  Print report ## prints the report in the window field
```

Send to clipboard

Command group	Flag affected	Reversible
Report destinations	NO	YES

Syntax

Send to clipboard

Description

This command sends the output of any subsequent reports to the clipboard. The report is printed as a text-only file and all text formatting is ignored. If two reports are sent to the clipboard, the second report overwrites the first. Once a report has been sent to the clipboard, you can launch another program, such as a word processor, and paste the report into it.

If you use **Send to clipboard** in a reversible block, the report destination reverts to its former setting when the method terminates. The contents of the clipboard are not altered by the command or its reversal.

If you want to copy pictures from a report to the clipboard, you can print the report to screen and use the mouse to select the area required. The standard Edit menu Copy option will copy the graphic to the clipboard.

Example

```
Send to clipboard
Set report name rMyReport
Print report
# now launch word processor and paste
Launch program NOTEPAD.EXE Returns lStatus
If lStatus
  Paste from clipboard
End If
```

Send to DDE channel

Command group	Flag affected	Reversible
Report destinations	NO	YES

Syntax

Send to DDE channel

Description

This command directs any subsequent reports to a DDE channel. The current channel is defined by Set DDE channel number. An error occurs if the channel is not open or if the report is not printed with an export format.

Each record within the report is prepared and sent by Omnis as the data in a Poke message. The term “Poke” is defined by the DDE protocol and refers to messages carrying data which set field values in the target program. The server’s item names, into which the exported data is read, are defined by Set DDE channel item name.

The subsequent print commands will send to the channel number which is current at the time of the print command, not at the time of the Send to DDE channel command.

If you use **Send to DDE channel** in a reversible block, the report destination reverts to its former setting when the method terminates.

It may be the case that an export format for a particular Omnis report does not correspond to any of the formats supported by DDE. If a mismatch occurs, there will be an error message at the Print report or Prepare for print command.

Example

```
Send to DDE channel
Set export format {Delimited (commas)}
Set report name rMyReport
Clear DDE channel item names
Set DDE channel item name {Name}
Set DDE channel item name {Telephone}
Print report
Close DDE channel
```

Send to file

Command group	Flag affected	Reversible
Report destinations	NO	YES

Syntax

Send to file

Description

This command directs the report output to the currently selected print file. The report is sent as a text file (no text style or formatting) with the appropriate line terminators. The print file is not closed when a report finishes so you can print multiple reports without changing the destination or the name of the print file.

When you select the destination using the dialog window (see Prompt for destination), the Page size pushbutton lets you set up the form feeds and lines per page. These settings are stored in the preferences file.

Set lines per page lets you specify page length from methods. If the Send form feed option is selected, the end of each page is marked by a form feed character; otherwise, the pages are forced by sending multiple line feeds. You use Set print or export file name to designate the file name.

If you use **Send to file** in a reversible block, the report destination reverts to its former setting when the method terminates.

Example

```
Send to file
Set lines per page {46}
Calculate lPrintFileName as con(sys(115),'myPrintedReport.txt')
Set print or export file name {[lPrintFileName]}
Print report
```

Send to page preview

Command group	Flag affected	Reversible
Report destinations	NO	YES

Syntax

Send to page preview ([*Do not wait for user*][,*Hide until complete*]) title[/*left/top/width/height/stk/cen/max*]

Options

Do not wait for user	Unless this option is specified, the user must close the window before method execution continues and before doing anything else
Hide until complete	If specified, the report window is not displayed until the report has been completely generated

Description

This command sends the report instance to a page preview screen. This lets the user check the final page layout before printing. On small screens, the text is Greeked, that is, each character is represented by a dot.

The *Do not wait for user option* allows subsequent method lines to execute or lets the user do other things without closing the report; the default is to gray out all menus while a screen report is displayed. You may want to have several reports on the screen for reference while doing some other work with the library. Without the option, the user must close the window before doing anything else.

The *Hide until Complete* option suppresses the output until all the report data is ready. Normally, you can view the first part of the report before all the records have been prepared.

Title and Position

You can give each page preview a title and control its position and size. The *Left/Top/Right/Bottom* values fix the positions of the four corners to screen pixel resolution. The */STK* parameter offsets the top left-hand corner from the last page preview and */CEN* positions the page preview in the middle of the screen.

The Page preview window can be opened maximized by specifying the /MAX parameter or in $windowprefs.

If you change the shape and size of the page preview window it will no longer reflect the paper size.

If you use **Send to page preview** in a reversible block, the report destination reverts to its former setting when the method terminates.

Example

```
# Example shows how to stack 2 page previews showing UK and US customers
Set report name rMyReport
Send to page preview (Do not wait for user) UK customers/STK
Set search as calculation {cCountry='UK'}
Print report (Use search,Do not finish others) {rInst1}
Send to page preview (Do not wait for user) USA customers/STK
Set search as calculation {cCountry='USA'}
Print report (Use search,Do not finish others) {rInst2}
```

Send to port

Command group	Flag affected	Reversible	Execute on client
Report destinations	NO	YES	NO

Syntax

Send to port

Description

This command directs the report output to the currently selected port. The report is sent as a stream of text with the appropriate line terminators. The port is selected with the Set port name command.

If you use **Send to port** in a reversible block, the report destination reverts to its former setting when the method terminates.

Example

```
If platform()='X'
  Set port name {2 (Printer port)} ## macOS
Else
  Set port name {COM2:} ## Windows & Linux
End If
Send to port
Set port parameters {9600,n,70,0}
Print report
```

Send to printer

Command group	Flag affected	Reversible
Report destinations	NO	YES

Syntax

Send to printer

Description

This command sends the report to the current printer. You can choose the printer using the Select printer command.

If you use **Send to printer** in a reversible block, the report destination reverts to its former setting when the method terminates.

Example

```
Set report name rMyReport
Send to printer
Print report
```

Send to trace log

Command group	Flag affected	Reversible	Execute on client
Debugger	NO	NO	YES

Syntax

Send to trace log ([*Diagnostic message*] [,*Always log*] [,*Start diagnostic logging*] [,*Stop diagnostic logging*]) *text*

Options

Diagnostic message	If specified, the message will only be added to the trace log, if the trace log has been set to log diagnostic messages
Always log	If specified, always log the message even if $nodebug is true for the library or the local debugger is disabled (this option is ignored for a diagnostic message)
Start diagnostic logging	If specified, the command switches on the Log Diagnostic Messages trace log option, before logging the message if the other command options allow. Also, if specified with an empty message to log, the command does not log an empty line
Stop diagnostic logging	If specified, the command switches off the Log Diagnostic Messages trace log option, after logging the message if the other command options allow. Also, if specified with an empty message to log, the command does not log an empty line

Description

This command sends a specified line of text to the trace log. The text can contain square bracket notation. You can use text styles (generated using the style() function inside square brackets) to (for example) apply colors to sections of the logged text when it is displayed in the trace log panel in the browser or the trace log window; such styles are stripped when writing the trace log line to the text log file in the logs folder. The trace log renders the text styles if the entry traceLogUsesStyles in the defaults section of config.json

is set to true. Note that if you use styles other than kEscColor and kEscStyle, these styles are ignored when copying selected trace log lines to the clipboard as HTML.

For JavaScript client-executed methods, this command sends the text to the JavaScript console (provided it is available). In this case, text styles are not supported.

Example

```
# send messages to the trace log
Open trace log (Clear trace log )
Send to trace log Current task is [$ctask().$name]
Send to trace log Current class is [$cclass().$name]
For lCount from 1 to 10 step 1
   Send to trace log Value lCount is [lCount]
End For
Send to trace log End of For Loop
```

Set 'About...' method

Command group	Flag affected	Reversible
Omnis environment	NO	YES

Syntax

Set 'About...' method [*name/*]*name*

Description

This command changes the "About..." option by calling the specified method which you should set to open a different About window. Omnis executes the specified method when this option is selected in exactly the same way as if it had been selected from a menu, for example, standard windows are closed. If you use **Set 'About...' method** in a reversible block, the command is reversed when the method terminates.

There are no restrictions on what you can do in the **Set 'About...' method**, that is, the method that is called. Extra care is needed to ensure that the method does not alter any variables, lists or the status of the flag.

Example

```
# Open the window wMyAbout instead of the standard Omnis about box
Set 'About...' method cMyCodeClass/AboutBox
# method AboutLibrary in code class cMyCodeClass
Open window instance wMyAbout
```

Set advise options

Command group	Flag affected	Reversible	Ex
Exchanging data	NO	YES	NO

Syntax

Set advise options ([*Find/next/previous*][,*OK*][,*Redraw*])

Options

Find/next/previous	If specified,Omnis will send DDE advise messages to the client application,when Find/Next/Previous or Clear commands are executed (see command Advise on find/next/previous)
OK	If specified,Omnis will send DDE advise messages to the client application,when an evOK event occurs (see command Advise on OK)
Redraw	If specified,Omnis will send DDE advise messages to the client application,when a redraw occurs (see command Advise on redraw)

Description

DDE command, Omnis as server. This command determines when Omnis is permitted to send requested Advise messages to the client application. When the Accept advise requests option is active, Omnis will accept Advise requests from the client program. By default, the client program will only be advised of the values requested from Omnis when Send advises now is executed.

However, S**et advise options** specifies other events which will cause the values to be sent. There are three checkbox options available for this command: *Find/next/previous*, OK, and Redraw.

The *Find/next/previous* option sends the requested Advise value whenever a Find/next/previous command or a Clear command is executed. The *OK* option sends the requested Advise value whenever an Enter Data or Prompted Find ends with an OK. The *Redraw* option sends the requested Advise value whenever a Redraw is executed.

Each of these options in **Set advise options** has its command equivalent within the Exchanging Data... group, whose function is identical. These commands are listed as Advise on Find/next/previous, Advise on OK, and Advise on redraw.

Example

```
Set server mode (Field requests,Advise requests)
Set advise options (Find/next/previous,OK)
OK message {Server mode for DDE enabled}
```

Set bottom margin

Command group	Flag affected	Reversible
Report parameters	NO	NO

Syntax

Set bottom margin ([*Measurement in cms*]) {measurement-in-inches/cms}

Options

Measurement in cms	If specified ,the measurement parameter is a value in centimetres rather than inches

Description

This command specifies the bottom margin for the current report class. It overrides the $bottommargin property until such time as the current report is reset.

Example

```
# Prompt user and set appropiate margins
Set report name rMyReport
Yes/No message {Print on metric A4 paper?}
If flag true
  Set bottom margin (Measurement in cms) {2.34}
  Set top margin (Measurement in cms) {1.2}
Else
  Set bottom margin {1.0}
  Set top margin {1.0}
  # Default measurement is inches
End If
Print report
Set report name rMyReport ## the settings for rMyReport are now reverted

# Alternatively, you can use notation to set the bottom margin
Do $clib.$reports.rMyReport.$bottommargin.$assign(1.0)
```

Set break calculation

Command group	Flag affected	Reversible	Execute on client
Debugger	NO	NO	NO

Syntax

Set break calculation on *field-name* {*calculation*}

Description

This command stops method execution when the specified calculation evaluates to true; all values except zero are considered true. You use **Set break calculation** after a Variable menu command: Set break on calculation {field-name} command. The field used in the command does not have to feature in the calculation but is used to "label" the break within Omnis.

At breakpoints, a method design window is opened with the current method loaded and the breakpoint command highlighted. You can examine field values by right button/ Ctrl-clicking on the field or step through the remaining method.

Setting up calculated breakpoints slows down method execution considerably so you should use them sparingly. In runtime the command does nothing.

Example

```
# pause method execution when lMyBoolean=kTrue
Calculate lMyBoolean as kFalse
Variable menu command : Set Break On Calculation {lMyBoolean}
Set break calculation on lMyBoolean {lMyBoolean=kTrue}
For lCount from 1 to 10 step 1
  If lCount=5
    Calculate lMyBoolean as kTrue
  End If
End For
```

Set class description

Command group	Flag affected	Reversible	Execute on client
Classes	YES	NO	NO

Syntax

Set class description {*class-name/description*}

Description

This command sets the description text for the specified library class. When a class is created, you must specify a class name and also an optional description of up to 255 characters. This command lets you set the description string for the specified library class. The original description for the specified class is cleared if the description parameter is left blank (or evaluates to an empty string). The flag is set if the description is changed.

Example

```
Calculate lString as 'My Class Description'
# set the class desciption to the contents of the local variable lString
Set class description {sMySearch/[lString]}
# show the new contents of the class description
OK message {[$clib.$classes.sMySearch.$desc]}
```

Set closed files

Command group	**Flag affected**	**Reversible**	**Execute on client**
Files	YES	YES	NO

Syntax

Set closed files {*list-of-files* (F1,F2,..,Fn)}

Description

This command sets the file mode of the specified file(s), other than a main file, to closed. Closing a file prevents any data from being read or changed in that file.

If you attempt to close the main file an error occurs. If you use Set closed files in a reversible block, the file mode is reset when the method terminates. **Set closed files** does not cancel the Prepare for update mode. In multi-user libraries, closing a file prevents Omnis from locking it.

Closing a parent file when editing a child has the effect of protecting the connections from child to parent from change and saves time when locating child records because the parent record is not loaded.

In the method editor, a list of files is displayed. You can Ctrl/Cmnd-click on the file names to select multiple names.

Example

Prevent data from being read or changed in the files

fAccounts and fInvoices

Set closed files {fAccounts,fInvoices}

Set current data file

Command group	Flag affected	Reversible	Execute on client
Data files	NO	YES	NO

Syntax

Set current data file {*internal-name*}

Description

This command sets the specified data file the "current" data file. If your methods refer to file class names without specifying the data file, it is essential to make the appropriate data file current before setting a main file.

Example

```
Open data file {Archive.df1/DataFileA}
Open data file (Do not close other dat) {myData.df1/DataFileB}
Set current data file {DataFileA}
Set main file {fCustomers}
# fCustomers.Field1 now refers to DataFileA.fCustomers.Field1
```

Set current list

Command group	Flag affected	Reversible	Execute on client
Lists	NO	YES	NO

Syntax

Set current list *list-or-row-name*

Deprecated Command

This command has been deprecated and is no longer visible in the Code Assistant in the Code Editor (it will not appear when you type the first few characters), although it is still present in Omnis Studio and will continue to function if used in legacy code. You can show this command by disabling the appropriate Command Filter in the **Modify** menu in the Code Editor.

Note that you can create an instance variable of List data type, and such lists do not need to be made "current" since they are instantiated automatically and made current in the context of the current method or instance.

Description

This command sets the current list, that is, the list to be processed in the subsequent list commands. You can make any type of list the current list, including local, class, and library variables of list data type. If you use this command as part of a reversible block, the current list reverts to its former value when the method containing the reversible block finishes.

See also Define list.

Example

```
Set current list iMyList
Define list {fCustomers}
Set main file {fCustomers}
Build list from file on fCustomers.CustomerID
```

Set DDE channel item name

Command group	Flag affected	Reversible	Ex
Exchanging data	YES	NO	NO

Syntax

Set DDE channel item name {*server-data-item-name*}

Description

DDE command, Omnis as client. This command specifies the server data item name to which you can send the exported report. When transmitting a Send to DDE channel report, Omnis takes the channel item name and uses it as the server item name which is to be sent.

The flag is cleared if the item name is too long, thus causing a memory allocation error to take place.

The item names set in the command accumulate over each use of the command until a Clear DDE channel item names is issued.

Within a client library, for example, a report class is created which sends the fields ClF1, ClF2...ClF5 to the current channel. At the server end of the conversation, the fields are to be read into five fields server1, server2...server5. Before you can print the report, the method must contain the following commands:

Example

```
Set report name rMyReport
Send to DDE channel
Set DDE channel number {1}
Open DDE channel {Omnis|myLibrary}
Send command {[[TakeControl]}
If flag true
  Set DDE channel item nam {server1}
  Set DDE channel item nam {server2}
  Set DDE channel item nam {server3}
  Set DDE channel item nam {server4}
  Set DDE channel item nam {server5}
  Print report
End If
```

Set DDE channel number

Command group	Flag affected	Reversible	Ex
Exchanging data	YES	YES	NO

Syntax

Set DDE channel number {*calculation*}

Description

DDE command, Omnis as client. This command sets the channel number to be used in subsequent DDE commands. Each channel number identifies a particular conversation.

The channels are numbered from 1 to 8, and the flag is cleared if an invalid channel number is used. If you omit the channel number, it defaults to 1. The channel number selected can be the result of a calculation. All subsequent channel commands function on the current channel number. To select another channel, you must use a new Set DDE channel number command.

Example

```
Set DDE channel number {2}
Open DDE channel {Omnis|Country}
If flag false
  OK message {The Country library is not running}
Else
  Send command {Do method Invoice}
  Do method TransferData
End If
```

Set default data file

Command group	Flag affected	Reversible	Execute on client
Data files	NO	YES	NO

Syntax

Set default data file {*list-of-files* (F1,F2,..,Fn)}

Description

This command sets the default data file to be the current data file. Normally, file classes are associated with whatever the current data file is, at the time of execution. You use Set current data file to change the identity of the current data file. As the current data file changes, the file classes are associated with the changed current data file.

Set default data file sets the data file, for the specified file class or list of file classes, to be fixed at whatever is the current data file at the time when the command executes. In other words, it creates an association between a list of file classes and the particular data file that was current. For these file classes, the data file becomes fixed (that is, the "default" data file) and does not change whenever the current data file changes. You can break the association with either a new **Set default data file** or a Floating default data file command.

When you close the default data file for a file, that file reverts to a floating state. This means that the default data file for that file reverts to the current data file and changes when the current data file changes.

Set default data file does not change the flag but is reversible, that is, when the command is reversed, the previous default data files are restored. A runtime error occurs if there are no data files open when the command is executed.

Example

```
Open data file {myDataFile} ## open first datafile
Open data file (Do not close other dat) {myOtherDataFile} ## open second datafile
Set default data file {fCustomers,fOrders}
Set current data file {myDataFile}
Set main file {fCustomers}
# This now refers to the myOtherDataFile NOT myDataFile which is the current data file
```

Set export format

Command group	Flag affected	Reversible
Report parameters	NO	NO

Syntax

Set export format {*export-format*}

Export Formats

Delimited (commas)
Delimited (tabs)
One field per line
Omnis data transfer
Delimited (user delimiter)

Description

This command specifies the export format to be used with the current report. The **Set export format** command lets you to override the parameters stored in the report class. You should use it after selecting a report class.

If you leave the name empty, the report is printed without an export format. An error occurs if the name is not a valid export format name. The name specified for the command can contain square bracket notation.

Translation

Export format names are not tokenized and therefore are not understood by foreign language versions of Omnis. To avoid this portability problem, you can always build a list of export formats and use the list to select a format (see the second Example below).

Example

```
# Ouput report rMyReport to a comma delemited file
Send to file
Set report name rMyReport
Set print or export file name {[con(sys(115),'output.txt')]}
Set export format {Delimited (commas)}
Print report
Close print or export file
# Set the export format to the second in the list iExportFormatList
Set current list iExportFormatList
Do iExportFormatList.$define(iExportFormat)
Build export format list
Do iExportFormatList.$line.$assign(2) ## Delimited (tabs)
Do iExportFormatList.$loadcols()
Set export format {[iExportFormat]}
```

Set file read-only attribute

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

Set file read-only attribute (*path*, *read-flag*) **Returns** *err-code*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This command lets you set the read-only attribute of the file specified in path-name. If you set the *read-flag* parameter to kTrue the file is set to read-only, or if kFalse the file is set to read/write.

It returns an error code (See Error Codes), or zero if no error occurs.

Example

```
# set the read-only attribute of lPathname to kTrue
Calculate lPathname as con(sys(115),'libraries',sys(9),'mylibrary.lbs')
Set file read-only attribute (lPathname,kTrue)
```

Set final line number

Command group	Flag affected	Reversible	Execute on client
List lines	YES	NO	NO

Syntax

Set final line number {*line-number* (*calculation*)}

Deprecated Command

This command has been deprecated and is no longer visible in the Code Assistant in the Code Editor (it will not appear when you type the first few characters), although it is still present in Omnis Studio and will continue to function if used in legacy code. You can show this command by disabling the appropriate Command Filter in the **Modify** menu in the Code Editor.

Description

This command explicitly sets the value of LIST.$linecount by specifying a line number or a calculation. Omnis expands or contracts any list as necessary and maintains the value of the LIST.$linecount property as the last line number. If the number of lines in the list is less than the number set for LIST.$linecount, Omnis adds empty lines to the end. If the number of lines is greater than LIST.$linecount, Omnis shortens the list and reduces the memory needed by the list.

You can use **Set final line number** to speed up list handling by setting the final line number to shorten lists, for example. The list is effectively cleared of data when the line number parameter is left blank (or evaluates to zero).

Example

```
# Reduce the number of lines in the list from 100 to 50
Set current list lMyList
Define list {lCol1}
For lCol1 from 1 to 100 step 1
  Add line to list {lCol1}
End For
OK message {List has [lMyList.$linecount] lines}
Set final line number {50}
OK message {List now has [lMyList.$linecount] lines}
```

Set import file name

Command group	Flag affected	Reversible	Execute on client
Importing and Exporting	YES	YES	NO

Syntax

Set import file name {*file-name*}

Description

This command specifies the name of the import file. The flag is set if the import file is successfully selected. You use the current import file in any subsequent Import field from file commands.

If you use **Set import file name** in a reversible block, the import file is closed when the method containing the reversible block terminates.

Example

```
# import from a csv file called myImport.txt in the root of your omnis tree
Calculate lImportPath as con(sys(115),'myImport.txt')
Set import file name {[lImportPath]}
Prepare for import from file {Delimited (commas)}
Import data lImportList
End import
Close import file
```

Set label width

Command group	Flag affected	Reversible
Report parameters	NO	NO

Syntax

Set label width ([Measurement in cms]) {measurement-in-inches/cms}

Options

Measurement in cms	If specified, the measurement parameter is a value in centimetres rather than inches

Description

This command specifies the width of the labels when printing labels. It overrides the value set in the report parameters dialog until the current report is next reset. The width is measured from the edge of one label to the corresponding edge of the next.

You can set up the vertical spacing between labels using Set record spacing.

Example

```
# Print labels with a width of 4.5 cms
Set report name rLabels
Set labels across page {4}
Set record spacing {3}
Set repeat factor {2} ## two of each label
Set label width (Measurement in cms) {4.5}
Print report ## default measurement is inches

# Alternatively, you can use notation to set the label width
Do $clib.$reports.rLabels.$labelwidth.$assign(4.5)
```

Set labels across page

Command group	Flag affected	Reversible
Report parameters	NO	NO

Syntax

Set labels across page {*number*}

Description

This command specifies the number of labels across the page for label printing. It overrides the setting in the report parameters dialog for the current report class. The setting remains in force until the next Set report name command.

When labels are printed, the vertical spacing from the top of one label to the next is set up using the $recordspacing property or from a method using Set record spacing.

Example

```
# Print 4 labels across a page
Set report name rLabels
Set labels across page {4}
Set record spacing {3}
Set label width {(Measurement in cms){4.5}}
Print report
```

Set left margin

Command group	Flag affected	Reversible
Report parameters	NO	NO

Syntax

Set left margin ([*Measurement in cms*]) {measurement-in-inches/cms}

Options

Measurement in cms	If specified, the measurement parameter is a value in centimeters rather than inches

Description

This command specifies the left margin for the current report class. It overrides the left margin setting in the report properties until such time as the current report is reset.

Example

```
# Prompt user and set appropiate margins
Set report name rMyReport
Yes/No message {Print on A4 paper?}
If flag true
  Set bottom margin (Measurement in cms) {2.34}
  Set top margin (Measurement in cms) {1.2}
  Set left margin (Measurement in cms) {1.2}
  Set right margin (Measurement in cms) {1.2}
Else
  # default measurement is inches
```

```
  Set bottom margin {0.5}
  Set top margin {0.5}
  Set left margin {0.5}
  Set right margin {0.5}
End If
Print report

# Alternatively, you can use notation to set the left margin
Do $clib.$reports.rMyReport.$leftmargin.$assign(0.5)
```

Set lines per page

Command group	Flag affected	Reversible
Report parameters	NO	NO

Syntax

Set lines per page ([*Send form feed*]) {*number*}

Options

Send form feed	If specified, Omnis outputs a form feed character at the end of each page of the report

Description

This command changes the number of lines per page for reports printed to file or port. You can send any report to a port or file using the Report destination dialog. When the destination is selected in this window, the number of lines is automatically set to the default number for the destination, so you must use **Set lines per page** after you have selected the report destination. The default lines per page setting is stored in the configuration file.

The *Send form feed* option lets you send a form feed character at the end of each page of the report; otherwise, multiple line feeds are sent.

Example

```
# Set the number of lines for each page of the report
# rMyReport to 66
Set report name rMyReport
Set lines per page (Send form feed) {66}
Print report
```

Set main file

Command group	Flag affected	Reversible	Execute on client
Files	NO	YES	NO

Syntax

Set main file {*file-name*}

Description

This command selects the "main file" class. **Set main file** is an essential command which you must execute before manipulating any data. You can insert or delete data only in the file designated as the main file. The designated file cannot be memory-only or closed.

The main file setting also determines which connected files are located when finding records with Find/Next/Previous, and which connections are updated. As each main file record is read, the connected records are automatically read in and made available for editing. When the main file is edited or inserted, all connections to its parent files are updated, unless the parent file is closed.

If Omnis attempts to execute a command which requires a main file before the main file is set, an error occurs. If the data file is not opened when the main file is set, Omnis will try to open the default data file and, if this is unsuccessful, will display the Change data file dialog box so that the user can select or create a data file.

Changing the main file after a Prepare for... command does not cancel Prepare for mode. When an update is encountered, the main file set at the time of the last Prepare for is used. (See Prepare for edit, Prepare for insert.)

If you use **Set main file** in a reversible block, the main file is reset to its previous value when the method containing the reversible block finishes.

Multiple open data files

If more than one data file is open, there is only one main file setting shared by all open data files. If you do not qualify a file class name with a data file, the current data file is assumed unless you have created an association between the file class and another data file using the Set default data file command.

Example

```
# Set the main file in a reversible block so it returns to
# it's former setting once this method terminates
Begin reversible block
  Set main file {fAccounts}
End reversible block
Prepare for insert
Calculate fAccounts.Code as 'AC01'
Calculate fAccounts.Surname as 'Smith'
Calculate fAccounts.Balance as 100
Update files
```

Set memory-only files

Command group	Flag affected	Reversible	Execute on client
Files	YES	YES	NO

Syntax

Set memory-only files {*list-of-files* (F1,F2,..,Fn)}

Description

This command sets the file mode of the specified file(s), other than the main file, to memory-only. You can use the fields from a memory-only file as global variables. To do this:

1. Create a file class with some fields of the required type (Character, Numeric, and so on).
2. Designate the file class as a memory-only file using this command.
3. Use the fields in your methods as temporary storage for data.

When a memory-only file is changed to read/write, its fields are not cleared from the current record buffer. Similarly, when a file is changed from read/write to memory-only, its records are not cleared. Memory-only fields are initialized as empty when the library is launched.

If used in a reversible block, **Set memory-only files** is reversed when the method containing the block finishes. This command does not clear the Prepare for update mode.

In the method editor, a list of files is displayed. You can Ctrl/Cmnd-click on the file names to select multiple names.

Example

```
# Use the fields in the file class fGlobals as temporary global variables
# which do not get written to a datafile
Set memory-only files {fGlobals}
Calculate fGlobals.gMyGlobalVar as 'My Global Var'
```

Set Omnis window title

Command group	Flag affected	Reversible
Omnis environment	NO	YES

Syntax

Set Omnis window title {*title*}

Description

This command changes the title on the Omnis application window (available under Windows and Linux only). The title parameter provides the new title which may contain square bracket notation. Unless reversed as part of a reversible block, the new title will remain until Omnis is restarted.

Example

```
# Set the Omnis window title to 'My Application'
Begin reversible block
  Set Omnis window title {My Application} ## for Windows/Linux only
End reversible block
```

Set page width

Command group	Flag affected	Reversible
Report parameters	NO	NO

Syntax

Set page width {*number*}

Description

This command changes the width of reports printed to file or port. The default setting is stored in the preferences file and is selected automatically when the destination is chosen. **Set page width** overrides this setting and must be used after selecting the report destination.

Example

```
# Set the page width for the report rMyReport to 45
Set report name rMyReport
Send to file
Set print or export file name {[con(sys(115),'output.txt')]}
Set lines per page {66}
Set page width {45}
Print report
Close print or export file
```

Set port name

Command group	Flag affected	Reversible
Report destinations	YES	NO

Syntax

Set port name {*port-name* (e.g. COMn: or LPTn:)}

Description

This command specifies the name of the port to be used with subsequent input or output via the port. The flag is set if the port is successfully selected. The command should follow Send to port. You can set the baud rate and other parameters for the port using Set port parameters.

Set port name is not reversible, but if you use it in a reversible block the specified port is closed when the method terminates.

If an error occurs, then this command sometimes generates a fatal error. You can use an error handler to intercept the fatal error; see Load error handler for details.

Example

```
Set report name rMyReport
Send to port
If platform()='X'
  Set port name {1 (Modem port)} ## macOS
Else
  Set port name {COM1:} ## Windows & Linux
End If
Set port parameters {1200,n,7,2}
Print report
```

Set port parameters

Command group	Flag affected	Reversible
Report destinations	YES	NO

Syntax

Set port parameters {*profile-spec* (<*profile*>,<*cpi*>,<*lpi*>) or parameters (e.g. 9600,n,8,1,x,10,6)}

Description

This command sets the port parameters. When you use *Select port* in a method, the baud rate and other parameters are set to the values configured for the system, or the values set by the last application to use the port. If you need to change the settings you can do so with this command, which should follow a Send to port. The flag is set if the command is successful.

The first five parameters only apply to serial ports. The last three (*CPI*, *LPI*, *timeout*) apply to both serial and parallel ports.

You specify the flow control parameter as either X, H or R (in upper or lower case).

- X means XON/XOFF protocol
- H means hardware handshaking, using both RTS/DTR and CTS/DSR.
- R is only available on Windows, and means hardware handshaking, using just RTS/DTR.

The maximum value for the transmit XON threshold and transmit XOFF threshold, in the flow control settings, is too large for Windows. 4096 works but in practice the value required will depend on what device is connected to the port.

The *CPI* and *LPI* parameters are numbers that specify characters and lines per inch. These are used by Omnis to justify fields in the report, and are not sent as control characters to the printer.

The *timeout* specifies the time in seconds that Omnis will wait for data transfer activity on the port, before aborting the transfer; each time there is new data transfer activity, Omnis restarts the timeout timer. If omitted, it defaults to the value stored in the report destination parameters for the port destination. A value of zero means that operations will not time out.

You can use a port profile name instead of the port parameters as described above.

Example

```
# example 1
# set a baud rate of 9600, no parity, eight data bits and 1 stop bit
Set port parameters {9600,n,8,1}

# example 2
# The extra comma indicates no change to the handshake parameter (X/H/R)
Set port parameters {9600,n,8,1,,10,6}
# set up the XON/XOFF handshale protocol
Set port parameters {9600,n,7,1,X}

# example 3
Set report name rMyReport
Send to port
If platform()='X'
  Set port name {1 (Modem port)} ## macOS
Else
  Set port name {COM1:} ## Windows & Linux
End If
Set port parameters {1200,n,7,2}
Print report
```

Set print or export file name

Command group	Flag affected	Reversible
Report destinations	YES	YES

Syntax

Set print or export file name {*file-name*}

Description

This command specifies the print file name to which printed output is to be directed. The flag is set if the print file is successfully selected. If you use **Set print or export file name** in a reversible block, the print file is closed when the method containing the reversible block terminates.

Set print or export file name closes the current print or export file, if any, and then opens the specified file. The property $root.$prefs.$appendfile determines how the file is opened. Either new data is appended to the current content, or the file is truncated to zero length.

Once the file name has been specified, Send to file directs the report output to the file. As each report is printed, its output is added to the end of the last report in the file.

If an error occurs, then this command sometimes generates a fatal error. You can use an error handler to intercept the fatal error; see Load error handler for details.

Example

```
If platform()='X'
 Set print or export file name {/Work/Output file2} ## macOS
Else
 Set port name {C:\work\output2.prn} ## Windows & Linux
End If
Send to file
Set report name rMyReport
Print report
Close print or export file
```

Set read-only files

Command group	Flag affected	Reversible	Execute on client
Files	YES	YES	NO

Syntax

Set read-only files {*list-of-files* (F1,F2,..,Fn)}

Description

This command sets the file mode of the specified file(s) to read-only. You can read but not write to a read-only file. **Set read-only files** does not cancel the Prepare for update mode.

If you use this command in a reversible block, the file reverts to its original mode when the method containing the command block terminates.

In multi-user systems, you use **Set read-only files** to prevent Omnis from locking certain files. When you make files read/write, they are locked and re-read. In multi-user systems, records such as invoice numbers and totals, accessed by a number of users, should be made read-only to prevent delays caused by record locking. You must return the file to read/write status momentarily while it is updated.

In the method editor, a list of files is displayed. You can Ctrl/Cmnd-click on the file names to select multiple names.

Example

```
# Data from fAccounts may be read, but not changed
Set read-only files {fAccounts}
Set main file {fInvoices}
Prepare for insert
Enter data
Update files if flag set
```

Set read/write files

Command group	Flag affected	Reversible	Execute on client
Files	YES	YES	NO

Syntax

Set read/write files {*list-of-files* (F1,F2,..,Fn)}

Description

This command sets the file mode of the specified file(s) to read/write. The read/write file mode is the default type of Omnis file; you can read and write data to a read/write file. The other three file modes are read-only, closed and memory-only. If a file is changed to read/write mode when in Prepare for update, the data for the file class is reread from disk. In multi-user systems, read/write files are locked when a Prepare for... command is executed.

The file mode will revert to its former state if you use the command in a reversible block.

In the method editor, a list of files is displayed. You can Ctrl/Cmnd-click on the file names to select multiple names.

Example

```
# Set the file fSequences to read/write so that we can get the next invoice number
Set read/write files {fSequences}
Set main file {fSequences}
Prepare for insert
Find first
Calculate fSequences.InvoiceNumber as fSequences.InvoiceNumber+1
Update files
Set read-only files {fSequences}
```

Set record spacing

Command group	Flag affected	Reversible
Report parameters	NO	NO

Syntax

Set record spacing ([*Measurement in cms*]) {measurement-in-inches/cms}

Options

Measurement in cms	If specified, the measurement parameter is a value in centimetres rather than inches

Description

This command specifies the line spacing for the record section of the current report class. It overrides the setting in the record section properties for the current report. The setting remains in force until the next Set report name.

Example

```
# Set the record spacing for the report rMyReport to 5.2 cms
Set report name rLabels
Set labels across page {3}
Set record spacing (Measurement in cms) {5.2} ## default is inches
Print report

# Alternatively, you can use notation to set the record spacing
Do $clib.$reports.rLabels.$recordspacing.$assign(5.2)
```

Set reference

Command group	Flag affected	Reversible	Execute on
Calculations	NO	NO	YES

Syntax

Set reference field-name **to** notation-or-calculation-for-an-item

Description

This command sets up and stores a reference to an item in a variable of type Item reference. It assigns an alias for an item of notation that you do not want to type each time the item is referenced in the code.

Note - for JavaScript client-executed methods this command is equivalent to Calculate.

Example

```
# declare local variable lRef of type Item reference in a window class
# set the local item reference variable lRef to a Balance field on a Page Pane
Set reference lRef to $cinst.$objs.PagePane.$objs.Balance
# now you can set the text color of the Balance field to red using lRef
Do lRef.$textcolor.$assign(kRed)
```

Set repeat factor

Command group	Flag affected	Reversible
Report parameters	NO	NO

Syntax

Set repeat factor {*number*}

Description

This command specifies the number of copies of the record section to be printed. It overrides the repeat factor specified in the report properties for the current report. **Set repeat factor** is particularly useful when printing multiple labels. The setting remains in force until the next Set report name. If the repeat factor is left blank (or evaluates to zero), the printing of the record sections of a report is suppressed completely; all heading sections, totals and subtotals are still calculated correctly.

Example

```
# Print 2 of each label
Set report name rLabels
Set labels across page {3}
Set repeat factor {2}
Set label width {3.4}
Print report

# Alternatively, you can use notation to set the repeat factor
Do $clib.$reports.rLabels.$repeatfactor.$assign(2)
```

Set report main file

Command group	Flag affected	Reversible
Report parameters	NO	NO

Syntax

Set report main file {*file-name*}

Description

This command specifies the main file for the current report. When a report is printed, Omnis uses the main file set by the last Set main file. **Set report main file** overrides the main file setting by specifying a new main file specifically for the report. The setting remains in force until the next Set report name.

Printing connected files

When printing connected files, it is essential that the child file is made the main file. Only the main file and its connected parent files are automatically read into the current record buffer.

If no sort fields are specified in the report class, the report generator steps through the records in the order defined by the record sequencing number for the main file. Sort fields let you reorder the report records.

Example

```
# Set the main file to fAccounts for the report rMyReport
Set report name rMyReport
Set report main file {fAccounts}
Clear sort fields
Set sort field fAccounts.Surname
Prompt for destination
Print report
# Alternatively, you can use notation to set the main file
Do $clib.$reports.rMyReport.$mainfile.$assign('fAccounts')
```

Set report main list

Command group	Flag affected	Reversible
Report parameters	NO	NO

Syntax

Set report main list *list-or-row-name*

Description

This command specifies a list as the source for the data for the current report. When a report is printed, Omnis uses the main file specified either in $mainfile or the file set by the last Set main file command. **Set report main list** lets you override the main file setting by specifying a list, from which data is read for the next printed report.

A list-based report prints one record for each line in the list. The data file is not used unless the report contains auto find fields. Sorting, searching, subtotals, and so on, continue to work the same way as for file-based reports. All field values are taken from the list and records are read in list order.

When a Prepare for print command is encountered, the current list or file setting overrides the Main file setting used in the report parameters dialog.

Example

```
# Set the main list for the report rMyReport
Set report name rMyReport
Set report main list tMyList
Prompt for destination
Print report
# Alternatively, you can use notation to set the main list
Do $clib.$reports.rMyReport.$mainlist.$assign('tMyList')
```

Set report name

Command group	Flag affected	Reversible
Reports and Printing	NO	YES

Syntax

Set report name *report-name*

Description

This command selects a report class for use with subsequent Print... commands. It terminates any report in progress.

If you use **Set report name** in a reversible block, the previous report name will be restored when the method terminates.

Example

```
# Print the report rMyReport to the selected
# destination
Prompt for destination
If flag true
  Set report name rMyReport
  Print report
End If
```

Set right margin

Command group	Flag affected	Reversible
Report parameters	NO	NO

Syntax

Set right margin ([*Measurement in cms*]) {measurement-in-inches/cms}

Options

Measurement in cms	If specified, the measurement parameter is a value in centimeters rather than inches

Description

This command specifies the right margin for the current report class. It overrides the right margin setting in the report properties until such time as the current report is reset.

Example

```
# Prompt user and set appropiate margins
Set report name rMyReport
Yes/No message {Print on A4 paper?}
If flag true
  Set bottom margin (Measurement in cms) {2.34}
  Set top margin (Measurement in cms) {1.2}
  Set left margin (Measurement in cms) {1.2}
  Set right margin (Measurement in cms) {1.2}
Else
  # default measurement is inches
  Set bottom margin {0.5}
  Set top margin {0.5}
  Set left margin {0.5}
  Set right margin {0.5}
End If
Print report
# Alternatively, you can use notation to set the right margin
Do $clib.$reports.rMyReport.$rightmargin.$assign(0.5)
```

Set search as calculation

Command group	Flag affected	Reversible	Execute on client
Searches	NO	YES	NO

Syntax

Set search as calculation {*calculation*}

Description

This command sets the current search as the single line calculation specified. The calculation replaces the current search class if one has been set. A subsequent report, Search list or a Find command with Use search will use the search calculation.

Search calculations allow the index optimization routine in Omnis to select a suitable index, provided that such an index is available. Leaving the calculation blank has the effect of clearing the previous search calculation.

Example

```
# Use Town index in fCustomers to find Londoners and then use search to locate Smiths.
Set main file {fCustomers}
Set search as calculation {fCustomers.Surname='Smith'}
Find on fCustomers.Town (Exact match,Use search) {'London'}
Do $cwind.$redraw()

# example 2, moves selected lines only between lists
```

```
Set current list lList2
Set search as calculation {#LSEL}
Merge list lList1 (Use search)
```

Set search name

Command group	Flag affected	Reversible	Execute on client
Searches	NO	YES	NO

Syntax

Set search name *search-name*

Description

This command sets the search class to be used with reports, Search list and Find (using search) commands. If no search class name is included, the current search is cleared. Search classes allow subsets of the records to be printed or worked on.

A Find first (Use search) command reads in the first record which matches the current search criterion and creates a find table. Subsequent Next commands print out the records in the table.

If used within a reversible block, the search name reverts to its former setting when the method terminates.

Example

```
# example 1
Set search name sArea
Set report name rMyReport
Print report (Use search)

# example 2
Set search name sArea
Set main file {fOrders}
Clear main & con
Prepare for print
# create table of records which match
Find first (Use search)
While flag true
  Print record
  Next
End While
End print
```

Set server mode

Command group	Flag affected	Reversible	Ex
Exchanging data	NO	YES	NO

Syntax

Set server mode ([*Field requests*][,*Field values*][,*Advise requests*][,*Commands*])

Options

Field requests	If specified, Omnis will accept DDE field request commands (see command Accept field requests)
Field values	If specified, Omnis will accept DDE field value commands (see command Accept field values)
Advise requests	If specified, Omnis will accept DDE advise request commands (see command Accept advise requests)
Commands	If specified, Omnis will accept DDE commands (see command Accept commands)

Description

This command sets Omnis to act as a DDE server and specifies which DDE commands it will accept. With one or more of the check box options selected Omnis will respond to the corresponding commands and demands from a client. If none is selected, server mode is deselected.

All four server mode check box options have equivalent DDE commands which are described separately: Accept field requests, Accept field values, Accept advise requests and Accept commands.

Irrespective of the mode selected, Omnis will only accept field values and commands when in enter data mode, and accept commands when no methods are running.

Omnis will only respond to a request to act as a server if the Initiate message from the client contains at least the name of the program, that is, Omnis. If the client specifies a topic, it has to be equal to the Omnis library name without the .lbr extension. Omnis responds with the current library name if the client does not specify the topic.

If no options are set, Omnis is disabled as a server except for the System Topic. If Omnis is already a server when the options under **Set server mode** are disabled, one of two things will happen:

1. If the options have been disabled during a reversible block, the client sending the Initiate message will get busy acknowledgments until the reversible command method finishes. You cannot initiate any new conversations during this time.
2. Omnis will end the communication by sending the client a Terminate message.

All four server mode options have equivalent commands which are described separately: Accept field requests, Accept field values, Accept advise requests and Accept commands.

Example

```
Set server mode (Field requests)
```

Set sort field

Command group	Flag affected	Reversible	Execute on clien
Sort fields	NO	YES	NO

Syntax

Set sort field field-name ([*Descending*][,*Upper case*][,*Subtotals*][,*New page*])

Options

Descending	If specified, data for the field is sorted in descending order
Upper case	If specified, data for the field is sorted in a case-insensitive manner, converting data to upper case before sorting
Subtotals	If specified, the Subtotal section in a report is printed when the value of the sort field changes

New page	Only relevant if you specify the subtotals option. If specified, Omnis starts a new page in a report when the value of the sort field changes

Description

This command specifies a field on which a list or report is to be sorted. The report generator systematically works through the records in the main and connected files and prints them using the report class definition. You can use sort fields to sort the records into a specific index order.

A report can be sorted on up to nine fields: you can specify sort fields in the report class or by using **Set sort field**. Since sort fields are cumulative, use Clear sort fields first to clear any that already exist.

When a report name is selected, the report class sort fields are used but you can override these sort fields by clearing them and specifying new sort ones with **Set sort field**. For nine sort fields, you use the **Set sort field** command nine times in succession. Using this method, however, can be slower than sorting on fields that are already indexed.

You can set the sort fields for lists using **Set sort field**. The Sort list command sorts the current list in the order specified by the current sort fields. Note that lists have to be explicitly redrawn before you can view the results of a sort.

If used within a reversible block, the sort field setting reverts when the method terminates.

The *Descending* option sorts the records in descending order. The *Upper Case* option converts lower case characters to upper case for the purpose of sorting. The *Subtotals* option causes the Subtotal section in the report to be printed when the value of the sort field changes. Thus, in the above example, when AREA changes, subtotals 1 is printed, when DEPT changes, subtotals 2 is printed, and so on. The *New Page* option starts a new page when the field value changes.

Example

```
# Sort the report on fields Surname,Balance
Set report name RCOMMISSION
Clear sort fields
Set sort field fAccounts.Surname
Set sort field fAccounts.Balance
Set report name rMyReport
Send to screen
Print report
```

Set timer method

Command group	Flag affected	Reversible	Execute on client
Methods	NO	YES	NO

Syntax

Set timer method *interval* (seconds) **sec** [*name*/]*name*

Description

This command calls the specified method at regular intervals while waiting for a keyboard input; the called method should preferably be one contained in a code class. You could use this command for automatic telephone dialing, regular checks for electronic mail, and so on.

The command specifies the timer method and the interval in seconds between calls to the timer method. This interval can be between 1 and 30,000 in the form “n sec” where n is the number of seconds. Omnis will start the next timer method when the method which is currently executing, finishes. Timer methods cannot operate in real time as Omnis will not execute a timer method while another method is running or when an OK or Yes/No message is displayed on the screen.

The timer method in your code class should not contain a Quit all methods as this will terminate any Enter data commands which are running. You can also use an Enter data inside a timer method: if so and you do not clear the timer method, the timer method continues to be active while Omnis carries out the Enter data part of the timer method.

You can use **Set timer method** in a reversible block, in which case the timer method is cleared when the executing method terminates.

Note that (from Studio 11 onwards) the timer method runs in the context of the current task, therefore you can access its task variables from the method. Note that the timer method will continue to run after the task closes.

Example

```
# Call the method Timer every 5 seconds
Set timer method 5 sec Timer
# method Timer
OK message {Timer method triggered}
```

Set top margin

Command group	Flag affected	Reversible
Report parameters	NO	NO

Syntax

Set top margin ([*Measurement in cms*]) {measurement-in-inches/cms}

Options

Measurement in cms	If specified, the measurement parameter is a value in centimeters rather than inches

Description

This command specifies the top margin for the current report class. It overrides $topmargin until such time as the current report is reset.

Example

```
# Prompt user and set appropiate margins
Set report name rMyReport
Yes/No message {Print on metric A4 paper?}
If flag true
  Set bottom margin (Measurement in cms) {2.34}
  Set top margin (Measurement in cms) {1.2}
Else
  Set bottom margin {1.0}
  Set top margin {1.0}
  # Default measurement is inches
End If
Print report
Set report name rMyReport ## the settings for rMyReport are now reverted
# Alternatively, you can use notation to set the top margin
Do $clib.$reports.rMyReport.$topmargin.$assign(1.0)
```

Set top window title

Command group	Flag affected	Reversible	Execute on client
Windows	NO	YES	NO

Syntax

Set top window title {*title*}

Description

This command specifies the title for the top window instance. You can use square bracket notation within the window title. The title of the top window instance is cleared if you omit the window title parameter (or it evaluates to an empty string). The title reverts to the normal title if the window instance is closed and reopened. An error occurs if there is no window instance.

If you use **Set top window title** in a reversible block, the title reverts to its normal value when the method containing the reversible block terminates.

Example

```
# Use the value of lName in the window title
Prompt for input Name ? Returns lName
Set top window title {Messages for [lName]}
```

Show 'About...' window

Command group	Flag affected	Reversible
Omnis environment	NO	NO

Syntax

Show 'About...' window

Description

This command displays the standard "About..." window which is available as an option in the Help menu under Windows and Linux, or the Apple menu under macOS. You can change the standard "About..." screen with the Set 'About...' method command.

Show docking area

Command group	Flag affected	Reversible	Execute on client
Toolbars	NO	NO	NO

Syntax

Show docking area ([*Show text*][,*Large Icons*][,*Variable Text Width*]) {*docking-area* (e.g. kDockingAreaBottom)}

Options

Show text	If specified, the docking area will display text for objects that have an associated text string
Large Icons	If specified, the docking area will display large icons (32x32) rather than small icons (16x16)

Variable Text Width	If specified, and 'Show text' is also specified, the docking area will adjust the width of the objects according to the width of the text

Description

This command opens the top, bottom, left, or right docking area into which toolbars may be installed. The docking area is specified using one of the constants: kDockingAreaTop, kDockingAreaBottom, kDockingAreaLeft, kDockingAreaRight or kDockingAreaFloating.

When a toolbar is created each control may have a text label, for example, a Print button may have the word "Print" associated with it. The *Show text* option allows these text labels to be shown beneath the buttons.

Example

```
Show docking area {kDockingAreaLeft}
Install toolbar {tbMyToolbar/kDockingAreaLeft}
# or you can use the notation
Do $root.$prefs.$dockingareas.$assign(kDockingAreaLeft)
Do $clib.$toolbars.tbMyToolbar.$open('*',kDockingAreaLeft) Returns lToolBarRef
```

Show fields

Command group	Flag affected	Reversible	Execute on client
Fields	NO	YES	NO

Syntax

Show fields {*list-of-field-names* (Name1,Name2,...)}

Description

This command shows the specified window field or list of fields. You can hide fields with Hide fields or using the notation. Inactive pushbuttons with the Do not gray attribute cannot be made visible with this or any other command.

If you use **Show fields** in a reversible block, the specified fields are hidden when the method containing the reversible block terminates.

Example

```
Yes/No message {Do you want to show fields?}
If flag true
  Begin reversible block
    Show fields {myField1,myField2}
  End reversible block
End If
# do something
Quit method
# now this method ends and the fields are re-hidden as they are in a reversible block
# To show a single field on the current window
Do $cwind.$objs.myField1.$visible.$assign(kTrue)
# to show all fields on the current window
Do $cwind.$objs.$sendall($ref.$visible.$assign(kTrue))
```

Show Omnis maximized

Command group	Flag affected	Reversible
Omnis environment	NO	NO

Syntax

Show Omnis maximized

Description

This command shows Omnis at its maximum size within the application window. This command performs the same action as the *Maximize* option in the System menu and the *Maximize* button on the application window.

Example

```
# Maximize Omnis after processing
Show Omnis minimized
For lCount from 1 to 100000 step 1
# delay
End For
Show Omnis maximized
```

Show Omnis minimized

Command group	Flag affected	Reversible
Omnis environment	NO	NO

Syntax

Show Omnis minimized

Description

This command minimizes Omnis which subsequently appears as an icon at the bottom of the screen.

Example

```
# Minimize Omnis while processing
Show Omnis minimized
For lCount from 1 to 100000 step 1
  # delay
End For
Show Omnis maximized
```

Show Omnis normal

Command group	Flag affected	Reversible
Omnis environment	NO	NO

Syntax

Show Omnis normal

Description

This command shows Omnis at its normal size within the application window. Icons for other applications are visible along the bottom of the screen.

Example

```
# Return Omnis to its original size after processing
Show Omnis minimized
For lCount from 1 to 100000 step 1
 # delay
End For
Show Omnis normal
```

Signal error

Command group	Flag affected	Reversible	Execut
Error handlers	NO	NO	NO

Syntax

Signal error {*error-number*, *error-text* (e.g. 5, 'Error 5 occurred')}

Description

This command reports a fatal error which can be either a user-defined error or a built-in Omnis error. A fatal error is any error that normally halts method execution and reports an error (for example, syntax error, or an out of memory error). Built-in Omnis errors are reported in #ERRCODE and #ERRTEXT, the latter is limited to 255 characters.

The fatal error is reported with the specified error code and text. Any error handler for that code will be invoked. If there is no error handler or the error handler does not make a set error action (SEA), the debugger is invoked, if available. Otherwise, execution halts with the error message.

This command is useful for trapping user-defined errors, and is a convenient tool for triggering an error situation inside Omnis for whatever condition you may want to specify.

Example

```
Test for only one user
If flag false
  Signal error {99,'Test for one user failed'}
End If
```

Single file find

Command group	Flag affected	Reversible	Execute on client	P
Finding data	YES	YES	NO	A

Syntax

Single file find on field-name ([*Exact match*]) {calculation}

Options

Exact match	If specified, the index value of the field in suitable records must equal the current value

Description

This command locates a record in a single file only. It is similar to the standard Find command but is not dependent on the main file; that is, the field used in **Single file find** does not have to belong to the main file and it does not read in the connected records. You can specify a calculation for Single file find which determines the value used in the Find. The *Exact match* option with a blank calculation indicates that the command is to be executed using the current value of the field, that is, the file is searched for a record whose index value matches the current value of the specified field.

In multi-user systems, a **Single file find** while in Prepare for... mode causes additional semaphores to be set. If the record is already locked, the user must wait for access to the record.

Example

```
# Find account lMyAccCode
Prompt for input Account Code ? Returns lMyAccCode (Cancel button)
Wait for semaphores
Single file find on fAccounts.Code (Exact match) {fAccounts.Code=lMyAccCode}
If flag false
  OK message {Can't find record}
End If
```

SMTPSend

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

SMTPSend (*server*,*from*,*to*,*subj*,*body*{Char|Bin|MIME-List}[,*cc*,*bcc*,*name*,*stsproc*,*pri*,*xtrahdrs*,*user*,*pass*,*secure* {Default zero insecure;1 secure;2 use STARTTLS},*verify* {Default kTrue}]) **Returns** *status*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

SMTPSend sends an Internet e-mail message via an SMTP server. It returns a Status value less than zero if an error occurs. Possible error codes are listed in the Web Command Error Codes Appendix.

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.server.com or 255.255.255.254. If the server is not using the default SMTP port (25 for non-secure connections, or 465 for secure connections), you can optionally append the port number on which the server is listening, using the syntax server:port, for example smtp.server.com:1234.

In addition, you can specify the argument of the HELO or EHLO command that will be sent to the server, via the Server parameter - pass the HELO/EHLO argument as a second string, separated from the server address or hostname by a comma, for example smtp.server.com:1234,www.mydomain.com. The argument of HELO or EHLO is typically the domain name of the sender.

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@omnis.net.

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@omnis.net. 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 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, content transfer encoding and content disposition 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. Also, ***SMTPSend*** assigns content disposition "attachment" to body parts with a file name. If you wish to override the default behavior for these three columns, 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.

User and *Pass* are required if **SMTPSend** is to use SMTP authentication when connecting to the server. If you omit the User parameter (or pass a zero length string) then the command does not attempt to use authentication when connecting to the server; otherwise, the command will use authentication when connecting to the server, where User is the user name and Pass is the password. Note that whereas SMTP supports many different forms of authentication, **SMTPSend** only supports the commonly used CRAM-MD5, LOGIN and PLAIN methods of authentication; if the server supports CRAM-MD5,then **SMTPSend** will use CRAM-MD5 if more than one of these three methods is available, as this is the most secure method of the three it supports.

Secure is an optional Boolean parameter which indicates if a secure connection is required to the server. Pass kTrue for a secure connection, in which case the built-in security technology will be used, so on Windows 'Secure Channel' (Schannel) is used, on macOS 'Secure Transport' is used, and on Linux OpenSSL is used.

SMTPSend also supports an alternative secure option, if you pass secure with the value 2, the connection is initially not secure, but after the initial exchange with the server, **SMTPSend** issues a STARTTLS SMTP command to make the connection secure if the server supports it (see RFC 3207 for details). Authentication occurs after a successful STARTTLS command.

Verify is an optional Boolean parameter which is only significant when *Secure* is not kFalse. When *Verify* is kTrue, the command instructs the installed SSL library to verify the server's identity using its certificate; if the verification fails, then the connection will not be established. You can pass *Verify* as kFalse, to turn off this verification; in this case, the connection will still be encrypted, but there is a chance the server is an impostor. In order to perform the verification, the installed SSL library uses the Certificate Authority Certificates in the cacerts sub-folder of the secure folder in the Omnis folder. If you use your own Certificate Authority to self-sign certificates, you can place its certificate in the cacerts folder, and the installed SSL library will use it after you restart Omnis.

Header Values Containing International Characters

SMTPSend supports RFC 2047, and uses it to encode international characters in header values, using UTF-8 as the character encoding.

Example

```
# Send email via the smtp server iOutServer from the email address iOutFrom to the email addresses in pToAddre
If pEnclosureList.$linecount>0
  # if the new e-mail contains enclosures, compose and send as multipart MIME content
  Do pEnclosureList.$redefine(lFileName,lFilePath)
  Do lMimeList.$define(lLevel,lContentType,lContentSubType,lFileName,
 lCharData,lBinData,lCharSet,lEncoding)
  Do lMimeList.$add(0,'multipart','mixed')
  Do lMimeList.$add(1,'text','plain',,pBody,,,)
  For lLineInList from 1 to pEnclosureList.$linecount step 1
```

```
    Do pEnclosureList.$line.$assign(lLineInList)
    Do pEnclosureList.$loadcols()
    Do lFileOps.$openfile(lFilePath)
    Do lFileOps.$readfile(lFileBinData)
    Do lFileOps.$closefile()
    Do lMimeList.$add(1,'application','octet-stream',lFileName,,lFileBinData,,)
  End For
  SMTPSend (iOutServer,iOutFrom,pToAddresslist,pSubject,lMimeList,
pCCAddresslist,pBCCAddresslist,iOutFromName,pStatusCall,pPriority)
Returns lStatus
Else
  # send message with no enclosures
  SMTPSend (iOutServer,iOutFrom,pToAddresslist,pSubject,pBody,
pCCAddresslist,pBCCAddresslist,iOutFromName,pStatusCall,pPriority)
Returns lStatus
End If
```

Sort list

Command group	Flag affected	Reversible	Execute on client
Lists	YES	NO	NO

Syntax

Sort list

Deprecated Command

This command has been deprecated and is no longer visible in the Code Assistant in the Code Editor (it will not appear when you type the first few characters), although it is still present in Omnis Studio and will continue to function if used in legacy code. You can show this command by disabling the appropriate Command Filter in the **Modify** menu in the Code Editor.

Description

This command sorts the current list in the order specified by the current sort fields. You can use Set sort field to set the sort fields. Note that lists have to be explicitly redrawn before you can view the results of a sort.

Example

```
Set current list iMyList
Define list {fCustomers}
Set main file {fCustomers}
Build list from file (Use search)
Clear sort fields
Set sort field fCustomers.Surname
Set sort field fCustomers.Town
Sort list
# or do it like this
Do iMyList.$sort(fCustomers.Surname,kTrue,fCustomers.Town,kTrue)
```

Sound bell

Command group	Flag affected	Reversible	Execute c
Message boxes	NO	NO	YES

Syntax

Sound bell

Description

This command sounds the system beep. You can sound the bell at any point in a method to draw attention to a particular method, field, message, error, and so on.

Example

```
# Sound the bell and open the window 'wMyErrorDialog'
Sound bell
Open window instance wMyErrorDialog
```

Split path name

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

Split path name (*path*, *drive-name*, *directory-name*, *file-name*, *file-extension*) **Returns** *err-code*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This command splits a full path name into its component parts: the drive name, directory and file name, and file extension. It returns an error code (See Error Codes), or zero if no error occurs. The following examples show how Split path name operates.

Windows

Path	Drive	Directory	Filename
C:\TESTDIR\TESTSDIR\TESTFILE	C:	\TESTDIR\TESTSDIR\	TESTFILE
C:\TESTDIR\TESTFILE.EXT	C:	\TESTDIR\	TESTFILE
C:\TESTFILE	C:	\	TESTFILE

Linux

Path	Drive	Directory	Filename	Exter
/TESTDIR/TESTSDIR/TESTFILE		/TESTDIR/TESTSDIR/	TESTFILE	
/TESTDIR/TESTFILE.EXT		/TESTDIR	TESTFILE	.EXT
/TESTFILE		/	TESTFILE	

Example

```
# split the path name lPathname
Calculate lPathname as 'c:\desktop\myfolder\mylibrary.lbs' ## Windows example using '\'
Split path name (lPathname,lDrive,lDirectory,lFileName,lExtension)
# lDrive= 'c:'
# lDirectory = '\desktop\myfolder\'
# lFileName = 'mylibrary'
# lExtension ='.lbs'
```

Sta:

Command group	Flag affected	Reversible
SQL Object Commands	NO	NO

Syntax

Sta: {*sql-script*}

Description

This command appends script data to the SQL buffer for the current method stack. The Begin statement command clears the buffer ready for a new statement.

The *sql-script* parameter is a text field which can contain square bracket and indirect square bracket notation (which you use to send data to the server as a "bind" variable): note the Method Editor will enclose the complete text in curly brackets automatically, so these do not need to be entered. A text editor will pop up when you enter or edit a **Sta:** line.

Multiple **Sta:** {*sql-script*} lines are added to the SQL buffer, and separated with the delimiter(s) specified by the Begin statement command. This allows you to split a SQL statement over more than one line, but note that literal values must not be split between lines. The SQL buffer can contain more than one SQL statement provided you use the appropriate statement delimiter.

Text loaded into the buffer must be valid SQL, and must be understood by the server. You can use square brackets to load the buffer with text obtained from Omnis functions, variables and calculations. Indirect bind variable notation of the form @[Field] is not evaluated in Omnis but is handled by the DAM, and lets you pass field values to the server without the need for them to be included in the text of a SQL statement.

You cannot insert an inline comment on any lines in a **Sta:** code block.

Example

```
# Open a multi-threaded omnis sql connection to
# the datafile mydatafile and create a statement to
# delete rows from the table Customers
Calculate lHostname as con(sys(115),'mydatafile.df1')
Do iSessObj.$logon(lHostname,'','','MYSESSION')
Do iSessObj.$newstatement('MyStatement') Returns lStatObj
Begin statement
Sta: {Delete From Customers}
Sta: {Where Cust_ID > 100}
Sta: {And Cust_ID<110}
End statement
Do lStatObj.$execdirect()
Do lStatObj.$fetch(lMyList,kFetchAll)
```

Standard menu command

Command group	Flag affected	Reversible	Execute on client
Menus	NO	NO	NO

Syntax

Standard menu command *command*

Description

This command performs the standard functionality of an option from one of the standard menus such as the File menu. This command can prove useful when defining a new menu class to replace a standard menu using Replace standard file menu or Replace standard edit menu.

Example

```
# Execute the 'Open Library' option from
# the standard edit menu
Standard menu command *File/11020 {Open Library...}
```

Start program maximized

Command group	Flag affected	Reversible
Operating system	YES	NO

Syntax

Start program maximized {*program-name*}

Description

This command starts up an application at its maximum screen size. The program name must be the pathname of the executable file. You can also specify the full pathname of a file, and other parameters, separated by a space from the program name. You can use this command on Windows and Linux, although on Linux the command does not maximize the application.

The flag is set if the program is found.

Example

```
# If the program lPath exists start it maximized
Calculate lPath as 'c:\program files\windows\accessories\wordpad.exe'
Test if file exists {[lPath]}
If flag true
  Start program maximized {[lPath]}
End If
```

Start program minimized

Command group	Flag affected	Reversible
Operating system	YES	NO

Syntax

Start program minimized {*program-name*}

Description

This command starts up an application as a minimized icon. The program name must be the pathname of the executable file. You can also specify the full pathname of a file, and other parameters, separated by a space from the program name. You can use this command on Windows and Linux, although on Linux the command does not minimize the application. The flag is set if the program is found.

Example

```
# If the program lPath exists start it minimized
Calculate lPath as 'c:\program files\windows\accessories\wordpad.exe'
Test if file exists {[lPath]}
If flag true
  Start program minimized {[lPath]}
End If
```

Start program normal

Command group	Flag affected	Reversible
Operating system	YES	NO

Syntax

Start program normal {*program-name*}

Description

This command starts up a Windows or Linux application at its normal screen size. The program name must be the pathname of the executable file. You can also specify the full pathname of a file, and other parameters, separated by a space from the program name.

The flag is set if the program is found.

Example

```
# If the program lPath exists start it in its normal screen size
Calculate lPath as 'c:\program files\windows\accessories\wordpad.exe'
Test if file exists {[lPath]}
If flag true
  Start program normal {[lPath]}
End If
```

Start server

Command group	Flag affected	Reversible	Execute on client
Threads	YES	NO	NO

Syntax

Start server {*stack-initialization-method* (parameters)}

Description

Start server starts the multi-threaded Web Client server. It creates the client method stacks and their associated threads, and starts the thread which listens for client requests. You use the property $root.$prefs.$serverstacks to specify the number of method stacks to be created.

Start server takes an optional *stack-initialization-method* and its parameter-list as parameters. If you specify the *stack-initialization-method*, **Start server** pushes this method on to every client method stack and allows it to execute, so that it can be used to initialize the state of the method stacks (so if $serverstacks is 5 it will execute five times).

The command clears the flag if it is used in a single threaded Omnis, or the serial number does not allow clients to connect. It generates a fatal error if for some other reason it is not possible to create the stacks and threads and start the listener.

Example

```
Start server
If flag false
  OK message {Failed to start multithreaded server}
End If
```

Stop server

Command group	Flag affected	Reversible	Execute on client
Threads	YES	NO	NO

Syntax

Stop server

Description

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

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

Stop server will fail and clear the flag, if you call it from a client method stack. In other words, you can only call **Stop server** from the main method stack.

Example

```
Stop server
If flag false
  OK message {Failed to stop multithreaded server}
End If
```

Swap lists

Command group	Flag affected	Reversible	Execute on client
Lists	YES	NO	NO

Syntax

Swap lists list-or-row-name

Deprecated Command

This command has been deprecated and is no longer visible in the Code Assistant in the Code Editor (it will not appear when you type the first few characters), although it is still present in Omnis Studio and will continue to function if used in legacy code. You can show this command by disabling the appropriate Command Filter in the **Modify** menu in the Code Editor.

Description

This command swaps the definition and contents of the specified list with that of the current list and sets the flag. After this command, the current list contains the fields and data which were held in the specified list, and the specified list contains the fields and data which were in the current list.

This command cannot be used to copy lists. To do this use Calculate LIST2 as LIST1.

Example

```
Set current list iList1
Define list {fCustomers}
Build list from file
Swap lists iList2
# Note: iList2 now contains the defintion and data from iList1 (the current list)
# iList1 is now empty
```

Swap selected and saved

Command group	Flag affected	Reversible	Execute on client
List lines	YES	NO	NO

Syntax

Swap selected and saved ([*All lines*]) {*line-number* (*calculation*)}

Options

All lines	If specified, the command affects all the lines in the list

Deprecated Command

This command has been deprecated and is no longer visible in the Code Assistant in the Code Editor (it will not appear when you type the first few characters), although it is still present in Omnis Studio and will continue to function if used in legacy code. You can show this command by disabling the appropriate Command Filter in the **Modify** menu in the Code Editor.

Description

This command swaps the Saved selection state and the Current selection state and sets the flag. To allow sophisticated manipulation of data via lists, a list can store two selection states for each line; the “Current” and the “Saved” selection. The Current and Saved selections have nothing to do with saving data on the disk; they are no more than labels for two sets of selections. The lists may be held in memory and never saved to disk: they will still have a Current and Saved selection state for each line but they will be lost if not saved. When a list is stored in the data file, both sets of selections are stored.

Swap selected and saved allows the Saved selection state of the specified line (or *All lines*) to be swapped with the Current set. You can specify a particular line in the list by entering either a number or a calculation. The *All lines* option swaps the selection status for all lines of the current list. The following example selects the middle line of the list:

Example

```
# Select all lines, save the selection, deselect all
# lines and then swap the selected and saved lines
# so all lines are selected
Set current list lMyList
Define list {lCol1}
For lCol1 from 1 to 6 step 1
  Add line to list {lCol1}
End For
Select list line(s) (All lines)
Save selection for line(s) (All lines)
Deselect list line(s) (All lines)
Swap selected and saved (All lines)
```

Switch

Command group	Flag affected	Reversible	Execute on client
Constructs	NO	NO	YES

Syntax

Switch expression

Description

This command initiates a **Switch** method construct. You use a **Switch** statement to select a course of action from a set of options based on the value of a variable, expression or calculation. It is similar to an If–Else If construct although the performance of a **Switch** construct tends to be faster.

The first line of the construction contains the **Switch** command. This defines the variable, expression or calculation on which the choice of action will depend. Following the **Switch** command, the Case commands provide values which, if matched with the expression supplied in the **Switch** line, cause the methods between case lines to be executed.

You can use the Break to end of switch command to jump out of the current Case statement and resume method execution after the *End Switch* command. Note you cannot use the *Break to end of loop* command to break out of a Switch construct.

You can nest multiple **Switch** statements, and embed other conditional statements such as If–Else constructs.

Example

```
# next button - only allow user to proceed to next page of a paged pane if the required information has been e
Calculate lPage as $cwind.$objs.PagedPane.$currentpage
Switch lPage
  Case 1
    If len(iSerialNumber)=0
      Calculate lErrorMsg as 'Please enter a serial number'
    End If
  Case 2
    If len(iUserName)=0
      Calculate lErrorMsg as 'Please enter your username'
    End If
  Case 3
    If iAgreeFlag=kFalse
      Calculate lErrorMsg as 'You may not proceed until you agree to the license agreement'
    End If
End Switch

If len(lErrorMsg)
  OK message {[lErrorMsg]}
Else
  Do $cwind.$objs.PagedPane.$currentpage.$assign(lPage+1) ## go to next page
End If
```

TCPAccept

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

TCPAccept (*socket*) **Returns** *socket*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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.

Example

```
# Accept incoming connections on port iPort
Calculate iPort as 6000
TCPSocket Returns iSocket
TCPBind (iSocket,iPort) Returns lStatus
TCPListen (iSocket) Returns lStatus
If lStatus=0
  Repeat
    TCPAccept (iSocket) Returns lConnectedSocket
  Until lConnectedSocket>=0
  # client connected, get the whole message sent
End If
TCPClose (iSocket) Returns lStatus
```

TCPAddr2Name

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

TCPAddr2Name (*address*) **Returns** *hostname*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded,allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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.

Example

```
# Return the Hostname for pIPAddress
TCPAddr2Name (pIPAddress) Returns lHostName
Quit method lHostName
```

TCPBind

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

TCPBind (*socket*,*service*|*port*) **Returns** *status*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# Get the next available socket and bind it to port iPort
Calculate iPort as 6000
TCPSocket Returns iSocket
TCPBind (iSocket,iPort) Returns lStatus
```

TCPBlock

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

TCPBlock (*socket*,*option* {Zero for blocking; Non-zero for non-blocking}) **Returns** *status*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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. Possible error codes are listed in the Web Command Error Codes Appendix.

Note:

If the connection is secure (see TCPConnect) then calls to TCPSend will always be blocking, even if the socket is marked as non-blocking.

Example

```
# Listen for incoming connections with blocking off
Calculate iPort as 6000
TCPSocket Returns iSocket
TCPBind (iSocket,iPort) Returns lStatus
TCPBlock (iSocket,1) Returns lStatus
TCPListen (iSocket) Returns lStatus
If lStatus=0
  Repeat
    TCPAccept (iSocket) Returns lConnectedSocket
  Until lConnectedSocket>=0
  # client connected
End If
TCPClose (iSocket) Returns lStatus
```

TCPClose

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

TCPClose (*socket*[,*option* {Default zero for complete;1 for partial;2 for abort}]) **Returns** *status*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# Close the socket bound to iPort
Calculate iPort as 6000
TCPSocket Returns iSocket
TCPBind (iSocket,iPort) Returns lStatus

TCPClose (iSocket)
```

TCPConnect

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

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

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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. Possible error codes are listed in the Web Command Error Codes Appendix.

Secure is an optional Boolean parameter which indicates if a secure connection is required to the server. Pass kTrue for a secure connection, in which case the built-in security technology will be used, so on Windows 'Secure Channel' (Schannel) is used, on macOS 'Secure Transport' is used, and on Linux OpenSSL is used.

Verify is an optional Boolean parameter which is only significant when Secure is not kFalse. When Verify is kTrue, the command instructs the installed SSL library to verify the server's identity using its certificate; if the verification fails, then the connection will

not be established. You can pass Verify as kFalse, to turn off this verification; in this case, the connection will still be encrypted, but there is a chance the server is an impostor. In order to perform the verification, the installed SSL library uses the Certificate Authority Certificates in the cacerts sub-folder of the secure folder in the Omnis folder. If you use your own Certificate Authority to self-sign certificates, you can place its certificate in the cacerts folder, and the installed SSL library will use it after you restart Omnis.

Notes:

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.

When using a secure connection all calls to TCPSend are blocking.

Example

```
# Connect to the server IP address iHostName on port iPort ready for
# sending a message
Calculate iHostName as '0.0.0.0'
Calculate iPort as 6000
TCPConnect (iHostName,iPort) Returns iSocket
If iSocket>0
 # connected
End If
```

TCPGetMyAddr

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

TCPGetMyAddr ([*socket* {Default 0}, *ipv6* {Default kFalse}]) **Returns** *address*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

TCPGetMyAddr is a domain name service command to resolve the IP address of the local computer running Omnis: the command may return a list of IP addresses in which case it uses the first address in the list. See **Additional Notes** below.

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.

- 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.
- 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

Possible error codes are listed in the Web Command Error Codes Appendix.

Additional Notes

When passing a socket to TCPGetMyAddr, the address returned is whatever the operating system API *getsockname* returns, and this can be either IP v4 or v6, which depends on how the connection was established. The ip v6 parameter to TCPGetMyAddr has no effect in this case.

When passing no socket to TCPGetMyAddr, the code uses the *gethostname* operating system API to obtain the name of the system, and then uses the *getaddrinfo* operating system API to obtain a list of addresses for the host. There can several addresses in the list returned, and TCPGetMyAddr uses the first address in the list that matches the request for ip v4 or v6.

Therefore, the information returned by this command is highly dependent on information exposed by operating system APIs, and these are only cross-platform in terms of their calling interface – the information they return depends on both the Operating System and the system configuration. In the case of *getaddrinfo*, the order of the returned items in the list is Operating system dependent.

Example

```
# Return the IP address of this machine
TCPGetMyAddr Returns lIPAddress
Quit method lIPAddress
```

TCPGetMyPort

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

TCPGetMyPort (socket) **Returns** *port*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# Bind iPort to iSocket and use TCPGetMyPort to return the
# port to which iSocket is bound in lMyPort.
Calculate iPort as 6000
TCPSocket Returns iSocket
TCPBind (iSocket,iPort) Returns lStatus
TCPGetMyPort (iSocket) Returns lMyPort
TCPClose (iSocket) Returns lStatus
```

TCPGetRemoteAddr

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

TCPGetRemoteAddr (socket) **Returns** *address*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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

Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# Listen for a incoming connections on port iPort and get the IP
# address iAddress of the remote computer
Calculate iPort as 6000
TCPSocket Returns iSocket
TCPBind (iSocket,iPort) Returns lStatus
TCPListen (iSocket) Returns lStatus
If lStatus=0
  Repeat
    TCPAccept (iSocket) Returns lConnectedSocket
  Until lConnectedSocket>=0
  TCPGetRemoteAddr (lConnectedSocket) Returns iAddress
End If
TCPClose (iSocket) Returns lStatus
```

TCPListen

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

TCPListen (*socket*) **Returns** *status*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# Listen for a incoming connections on port iPort
Calculate iPort as 6000
TCPSocket Returns iSocket
TCPBind (iSocket,iPort) Returns lStatus
TCPListen (iSocket) Returns lStatus
If lStatus=0
  Repeat
    TCPAccept (iSocket) Returns lConnectedSocket
  Until lConnectedSocket>=0
  # client connected
End If
TCPClose (iSocket) Returns lStatus
```

TCPName2Addr

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

TCPName2Addr (*hostname*[,*ipv6* {Default kFalse}]) **Returns** *address*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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.

Example

```
# Return the IP address for pHostName
TCPName2Addr (pHostName) Returns lIPAddress
Quit method lIPAddress
```

TCPPing

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

TCPPing (*hostname*[,*size*,*timeout*]) **Returns** *milliseconds*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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.

Example

```
# Ping iHostName to see if it is available
Calculate iHostName as '0.0.0.0'
TCPPing (iHostName) Returns iMilliseconds
If iMilliseconds<0
  If iMilliseconds=-1
    OK message {Timeout}
  Else
    OK message {Error [iMilliseconds]}
  End If
End If
```

TCPReceive

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

TCPReceive (*socket*,*buffer*[,*maxbytes*]) **Returns** *received-byte-count*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

TCPReceive receives data on a connected socket.

Socket is a long integer field containing the socket number of a connected socket.

Buffer is a character or binary field into which **TCPReceive** places the received data. If the field is character, then the response must be encoded in UTF-8; in this case, **TCPReceive** converts the received data from UTF-8 to character.

Maxbytes is an optional parameter which indicates the maximum number of bytes to be received. If you omit this parameter the command receives available data with no practical limit.

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

Notes

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 bytes 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 bytes returned to make sure there was no error.

Using **TCPReceive** to receive into a character field will not produce sensible results if the end of the received data stops part way through a UTF-8 encoded character.

Example

```
# Listen for incoming connections, if a connection is made get the message sent
Calculate iPort as 6000
TCPSocket Returns iSocket
TCPBind (iSocket,iPort) Returns lStatus
TCPListen (iSocket) Returns lStatus
If lStatus=0
  Repeat
    TCPAccept (iSocket) Returns lConnectedSocket
  Until lConnectedSocket>=0
  # client connected, get the whole message sent
  Calculate lMessage as ''
  Repeat
    Calculate lBuffer as ''
    TCPReceive (iSocket,lBuffer) Returns lMessageLength
    Calculate lMessage as con(lMessage,lBuffer)
  Until lMessageLength<=0
End If
TCPClose (iSocket) Returns lStatus
```

TCPSend

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

TCPSend (*socket*,*buffer*) **Returns** *sent-byte-count*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

Socket is a long integer field containing the socket number of a connected socket.

Buffer is a character or binary field containing the data to send on the socket. If you pass a character field, then **TCPSend** will convert the data to UTF-8, and then send the UTF-8.

TCPSend returns the number of bytes it sent to *sent-byte-count*, a long Integer field.

If the socket is in blocking mode, **TCPSend** always sends all of the data, unless an error occurs.

If the socket is in non-blocking mode, **TCPSend** sends as much data as it can without blocking.

If an error occurs, **TCPSend** returns a negative error code

Notes

If the connection is secure (see TCPConnect) then the send will always be blocking, even if the socket is marked as non-blocking.

Non-blocking sockets return an error code of -10035 if the socket cannot accept the data to send immediately. Some implementations of socket libraries may have limits on the number of bytes 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 *sent-byte-count* to determine how much of the buffer has actually been sent; if the value is less than the buffer size, you need to call **TCPSend** again, to send the rest of the buffer.

It does not make sense to send a character field on a non-blocking socket, because the *sent-byte-count* corresponds to the sent UTF-8 bytes.

Example

```
# Connect to the server IP address iHostName on port iPort and send the message iMessage
Calculate iHostName as '0.0.0.0'
Calculate iPort as 6000
Calculate lMessage as 'Hello remote application'
TCPConnect (iHostName,iPort) Returns iSocket
If iSocket>0
  # connected
  TCPSend (iSocket,lMessage) Returns lByteCount
End If
```

TCPSocket

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

TCPSocket () **Returns** *socket*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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.

Example

```
# Create a new socket, bind it to port 6000 and listen for an incoming client connection
Calculate iPort as 6000
TCPSocket Returns iSocket
TCPBind (iSocket,iPort) Returns lStatus
TCPListen (iSocket) Returns lStatus
If lStatus=0
  Repeat
    TCPAccept (iSocket) Returns lConnectedSocket
```

```
  Until lConnectedSocket>=0
  #; client con
End If
TCPClose (iSocket) Returns lStatus
```

Test check data log

Command group	Flag affected	Reversible	Ex
Data management	YES	NO	NO

Syntax

Test check data log ([Perform repairs])

Options

Perform repairs	If selected, repairs to the data file are automatically carried out

Description

This command tests if there are any reports of nonrepaired damage in the check data log. If the *Perform repairs* option is not specified, the flag is set if there are any reports of non-repaired damage.

If the *Perform repairs* option is specified, an attempt is made to repair the damage. There is no need for the check data log to be open. Furthermore, Omnis automatically tests that only one user is logged onto the data file (if not, the command fails with flag false), and further users are prevented from logging onto the data until the command completes.

If a working message with a count is open while the command is executing, the count will be incremented at regular intervals. The command may take a long time to execute, and it is not possible to cancel execution even if a working message with cancel box is open.

The command sets the flag if it completes the data repair successfully and clears the flag otherwise. The command is not reversible.

Example

```
Quick check
Test check data log
If flag true
  OK message {Problems found in data file}
  Open check data log
End If
```

Test clipboard

Command group	Flag affected	Reversible	Execute on client
Clipboard	YES	NO	NO

Syntax

Test clipboard field-name

Description

This command tests whether the data on the clipboard is suitable for pasting into the specified field or current selection. The command sets the flag to true if and only if there is data on the clipboard "suitable" for pasting into the specified or current field. "Suitability" here is defined by the standard type conversion built into Omnis, that is, a text field has to be presented with some text, and a picture field with something that can be handled as a picture, for example, a bitmap, metafile, PICT, OLE object, and so on.

Example

```
Test clipboard iPicture
If flag true
  Paste from clipboard iPicture (Redraw field)
End If
```

Test data with search class

Command group	Flag affected	Reversible	Execute on client
Searches	YES	NO	NO

Syntax

Test data with search class

Description

This command tests the record in the CRB against the current search class. It sets the flag if the record passes the test or if there is no current search class. If the data does not fit the current search class, the flag is cleared.

Test data with search class uses the current search as the condition of the test which has been set using Set search name or Set search as calculation.

Example

```
Calculate lCode as 'RT'
Set search as calculation {len(lCode)>2}
Test data with search class
If flag false
  OK message {Test failed, [lCode] invalid}
End If
```

Test for a current record

Command group	Flag affected	Reversible	Execute on
Finding data	YES	NO	NO

Syntax

Test for a current record {file-name}

Description

This command tests for the presence of a current record from a specified file class. The flag is set if a current record for the file is found and cleared if not. The flag is also cleared if the selected file is a memory-only or a closed file. The test is carried out on the main file if no other file class is specified.

Example

```
# If a record for fAccounts does not exist in the current record buffer, get the first
Set main file {fAccounts}
Test for a current record {fAccounts}
If flag false
  Find first
End If
```

Test for a unique index value

Command group	Flag affected	Reversible	Execute on
Finding data	YES	NO	NO

Syntax

Test for a unique index value on field-name

Description

This command tests the specified indexed field for a unique value. The flag is set if the current field value is a unique index value, and cleared if the value duplicates an existing index value. In a multi-user situation, no account is made of field values in records held by other work stations which are not yet updated to disk.

You use **Test for a unique index value** before storing a new value in a file. In the following example, the proposed new part number is tested against the existing file.

Example

```
# Insert account AC05 if it does not already exist
Set main file {fAccounts}
Prepare for insert
Calculate fAccounts.Code as 'AC05'
Test for a unique index value on fAccounts.Code
If flag true
  Update files
End If
```

Test for field enabled

Command group	Flag affected	Reversible	Execute on client
Fields	YES	NO	NO

Syntax

Test for field enabled {*field-name*}

Description

This command tests if the specified field on the top window instance is enabled, that is, if it is not currently disabled with Disable fields or by setting $enabled to kFalse. The flag is always cleared if there are no window instances open or if the field does not exist.

Example

```
Test for field enabled {myField}
If flag true
  Disable fields {myField}
Else
  Enable fields {myField}
End If
# or do it like this
If $cwind.$objs.myField.$enabled
  Do $cwind.$objs.myField.$enabled.$assign(kFalse)
Else
  Do $cwind.$objs.myField.$enabled.$assign(kTrue)
End If
```

Test for field visible

Command group	Flag affected	Reversible	Execute on client
Fields	YES	NO	NO

Syntax

Test for field visible {*field-name*}

Description

This command tests whether a particular field is visible. If the specified field in the top window instance is visible, that is, $visible is kTrue and the field has not been hidden with Hide fields, the flag is set. A field under another field or beyond the edge of the screen may be reported as visible and the flag set. The flag is always cleared if there are no window instances open or if the field does not exist.

Example

```
Test for field visible {myField}
If flag true
  Hide fields {myField}
Else
  Show fields {myField}
End If
# or do it like this
If $cwind.$objs.myField.$visible
  Do $cwind.$objs.myField.$visible.$assign(kFalse)
Else
  Do $cwind.$objs.myField.$visible.$assign(kTrue)
End If
```

Test for menu installed

Command group	Flag affected	Reversible	Execute on client
Menus	YES	NO	NO

Syntax

Test for menu installed {*menu-instance-name*}

Description

This command tests whether the specified menu instance is installed on the menu bar. The flag is set if the menu instance is on the menu bar and cleared if it is not, regardless of whether the menu instance is enabled or grayed out. The command does not apply to hierarchical and popup menus.

Example

```
# Install the menu mMyMenu if it is not already installed
Test for menu installed {mMyMenu}
If flag false
  Install menu mMyMenu
End If
```

Test for menu line checked

Command group	Flag affected	Reversible	Execute on client
Menus	YES	NO	NO

Syntax

Test for menu line checked *line* or *instance-name/line*

Description

This command tests whether the specified line of a menu instance is checked. You specify the *menu-instance-name* and the *line-number* of the menu line you want to test. The flag is set if the specified line of the menu instance is checked, and cleared if the line is not checked. The flag is always cleared if the menu instance is not installed on the menu bar.

You can check menu lines using Check menu line. Uncheck menu line removes the check.

Example

```
# Uncheck the menu line 'Large' if it is currently checked
Install menu mView
Check menu line mView/Large
Test for menu line checked mView/Large
If flag true
  Uncheck menu line mView/Large
End If
# Alternatively, you can see if a menu line is checked using notation
If $imenus.mView.$objs.Large.$checked
  Do $imenus.mView.$objs.Large.$checked.$assign(kFalse)
End If
```

Test for menu line enabled

Command group	Flag affected	Reversible	Execute on client
Menus	YES	NO	NO

Syntax

Test for menu line enabled *line* or *instance-name/line*

Description

This command tests whether the specified line of a menu instance is enabled. You specify the *menu-instance-name* and the *method-number* of the menu line you want to test. It sets the flag if the specified line of the menu instance is enabled. The flag is cleared if the menu instance is not installed on the menu bar.

This command may still return false if the current user has no access to the menu line or if the line is disabled because there is no current record, even after Enable menu line has been executed.

You can disable or enable menus using Disable menu line and Enable menu line.

Example

```
# Install the menu mMyMenu if it is not already installed
Test for menu installed {mMyMenu}
If flag false
  Install menu mMyMenu
End If
```

Test for only one user

Command group	Flag affected	Reversible	Execute c
Changing data	YES	NO	NO

Syntax

Test for only one user ([*All data files*])

Options

All data files	If specified, all data files are tested, rather than just the current data file

Description

This command tests whether the current data file is being used by a single user, and if so sets the flag.

If the *All data files* check box option is selected, all open data files are tested for a single user. The flag is cleared if any one data file has more than one user.

If the flag is set, further workstations are prevented from logging on to the tested data file(s) until the method containing the test command is terminated. The workstations will see a padlock cursor until the method terminates.

Omnis always sets the flag if the program is running in single user mode. Under Windows, this means that the data is on a DOS volume without the SHARE command having been run.

Example

```
Test for only one user
If flag false
  OK message {Sorry, option not allowed}
  Quit method kFalse
End If
Do method Invoices/InsertNew
```

Test for program open

Command group	Flag affected	Reversible	Execute on client	P
Operating system	YES	NO	NO	W

Syntax

Test for program open {*program-name*}

Description

This command tests whether the specified program is running. The flag is set if the specified program is running. You can use this command under Windows and Linux.

The program name can be the Windows module name, or the full pathname for the program. Under Windows NT/2000, the file PSAPI.DLL must be present in the Omnis directory or on the Windows path for this command to work. PSAPI.DLL is supplied in the Omnis directory of the Windows NT/2000 version of Omnis Studio.

Example

```
# Test to see if the program lPath is open
Calculate lPath as 'c:\program files\windows\accessories\wordpad.exe'
Test for program open {[lPath]}
If flag false
  Start program normal {[lPath]}
End If
```

Test for valid calculation

Command group	Flag affected	Reversible	Execute on c
Calculations	YES	NO	NO

Syntax

Test for valid calculation {*calculation*}

Description

This command lets you test a *calculation* before it is evaluated. It is essential to test strings to be evaluated by the eval(), evalf() and fld() functions before doing the evaluation. The flag is set to kTrue if the calculation is valid.

Example

```
Calculate lCalculation as 'lBalance < 0'
Test for valid calculation {evalf(lCalculation)}
If flag true
  Do lAccountsList.$search(evalf(lCalculation))
End If
```

Test for window open

Command group	Flag affected	Reversible	Execute on client
Windows	YES	NO	NO

Syntax

Test for window open {*window-instance-name*}

Description

This command tests if the specified window instance is open. If the window instance is open, Omnis sets the flag, otherwise the flag is cleared. Window instances are opened with Open window instance or the $open() method.

Example

```
# Open the window wMyWindow if it is not already open
Test for window open {wMyWindow}
If flag false
  Open window instance wMyWindow
End If
```

Test if file exists

Command group	Flag affected	Reversible
Operating system	YES	NO

Syntax

Test if file exists {*file-name*}

Description

This command tests if the specified file exists and can be opened. The flag is set if the file exists and can be opened. Otherwise, it is cleared. You can use this command to prevent the user from overwriting existing files with print files, and so on. To perform the test, the command opens the file in shared read mode, and then closes it, if the open was successful.

To just test for the existence of a file, without opening it, use the Fileops function FileOps.$doesfileexist.

You cannot use this command to check for the existence of a data file if the data file is in use by another workstation. Use Open data file for this type of checking.

You can use this command to test for the existence of a data file that is to be accessed using the ODB (Omnis Data Bridge). Specify the location of the file using the ODB syntax:

```
odb://[*address*:*port*:]*name*
```

where *address*:*port* is the TCP/IP address and port number of the ODB server, e.g. 127.0.0.1:5900, and *name* is the name of a data file accessed using the ODB server. You can omit *address*:*port*:, in which case Omnis uses the address and port stored in the $odbserver root preference. Note that the value of $odbserver is stored in the file odb.txt in the studio folder of the Omnis installation tree.

Example

```
# If the file myfile already exists in the root of the studio tree show a ok message
Calculate lPath as con(sys(115),'myfile')
Test if file exists {[lPath]}
If flag true
  OK message {The file [lPath] already exists}
Else
  Create file ([lPath])
End If
```

Test if list line selected

Command group	Flag affected	Reversible	Execute on client
List lines	YES	NO	NO

Syntax

Test if list line selected {*line-number* (*calculation*)}

Deprecated Command

This command has been deprecated and is no longer visible in the Code Assistant in the Code Editor (it will not appear when you type the first few characters), although it is still present in Omnis Studio and will continue to function if used in legacy code. You can show this command by disabling the appropriate Command Filter in the **Modify** menu in the Code Editor.

Description

This command tests the specified line of the current list and sets the flag if it is selected. You can specify a particular line in the list by entering either a number or a calculation. If the number is not specified, the test is performed on the current line of the list, that is, the line number held in LIST.$line.

Example

```
# If line 2 is selected show a message dialog
Set current list lMyList
Define list {lName,lBalance}
Add line to list {('Fred',100)}
Add line to list {('George',0)}
Add line to list {('Harry',50)}
Select list line(s) {2}
Test if list line selected {2}
If flag true
  OK message {List line 2 is selected}
End If
# Alternatively, you can check the $selected property
If lMyList.2.$selected
 OK message {List line 2 is selected}
End If
```

Test if running in background

Command group	Flag affected	Reversible
Omnis environment	YES	NO

Syntax

Test if running in background

Description

This command tests if Omnis is running in the background, that is, it sets the flag if Omnis is not the top application window.

Windows, macOS, and Linux all provide multi-tasking facilities. When another program is running, with Omnis in the background, you can continue with tasks such as importing data although the processor's time becomes shared between the current tasks. You can use this test to alter the behavior of the library when it becomes the background task.

Example

```
# Bring Omnis back to the front when another application goes to top
Show Omnis minimized

Calculate #F as kFalse
While flag false
  Test if running in background
End While
Show Omnis normal
```

Text:

Command group	Flag affected	Reversible	Execute on client
Text	NO	NO	YES

Syntax

Text: {text} ([*Carriage return*][,*Linefeed*][,*Platform newline*])

Options	
Carriage return	If specified, the command appends a carriage return, after it appends the text
Linefeed	If specified, the command appends a line feed, after it appends the text
Platform newline	If specified, the command appends the newline character sequence for the currently executing platform after it appends the text

Description

This command adds text to the text buffer for the current method stack: note the Method Editor will enclose the complete text in curly brackets automatically, so these do not need to be entered.

The **Text:** command supports leading and trailing spaces and can contain square bracket notation, that is, you can include or add the contents of a variable to the text buffer. You build up the text block using the Begin text block and one or more **Text:** commands. A text editor will pop up when you enter or edit a **Text:** line.

The *Carriage return*, *Linefeed*, and *Platform newline* options add the appropriate character(s) to the end of the current **Text:** line. When you have placed one **Text:** line and you press Ctrl/Cmnd-N to create a new method line, the **Text:** command is selected and the current carriage return and line feed options are copied to the new method line automatically. You should end a block of text with the End text block command, and you can return the contents of the text buffer using the Get text block command.

You cannot insert an inline comment on any lines in a **Text:** code block.

You cannot insert “Text:(” because the Text: command can have options (unlike Line:). For the Text: command, when you type (at the end of the code editor line, Omnis treats this as the start of the options, and the code assistant can then pop up the possible options. As soon as you type another character after the (, Omnis treats this as text comprising (followed by the character. This restriction is the best compromise that allows the code assistant to present the possible options, and allows text starting with (to be entered. As a workaround, you can enter [“(”] if you want to add an open parenthesis to the text.

Trace off

Command group	Flag affected	Reversible	Execute on client
Debugger	NO	NO	NO

Syntax

Trace off

Description

This command turns off the trace mode at a point in a method. See Trace on for more information about trace mode and using the debugger.

Example

```
Open trace log
#; the following lines are sent to the trace log ...
Trace on (Clear trace log )
For lCount from 1 to 5 step 1
  OK message {Sent to trace log}
End For
Trace off
# ...and the following line is not
OK message {Not sent to trace log}
```

Trace on

Command group	Flag affected	Reversible	Execute on client
Debugger	NO	NO	NO

Syntax

Trace on ([*Clear trace log]*)

Options

Clear trace log	If specified, the command clears the trace log

Description

This command sends all subsequent commands to the trace log and displays the current command in the method editor. It lets you turn on trace mode at a point in a method where you suspect that there may be a problem, or some code which is difficult to follow. In trace mode, the topmost method design window is continually changed to show the command being executed. Also when in trace mode, a trace log is maintained; this contains the class name and method name in the Item column and the command line text in the Data column, for all methods which are executed in trace mode or single-stepped. Error messages, breakpoints, and so on, which occur in trace mode are also entered in the trace log. The *Clear trace log* option deletes all existing entries before new lines are added to the log.

The trace log window is opened and brought to top either via the Tools menu or by the Open trace log command. This window allows the trace log to be viewed, cleared or printed, and lets you alter the maximum number of lines in the log. Double-clicking on a line in the trace log causes a method design window to be opened or brought to the top with the appropriate command displayed. If Shift is pressed when double-clicking, a new method design window is opened in preference to changing the identity of the class displayed in the existing method design window.

If the double-clicked line in the log is a field value line, the value window for that field is opened. The trace log is not adjusted when methods are modified. This means that trace log lines may point to the wrong command or no command if the class containing that method has been modified.

Example

```
Open trace log
# the following lines are sent to the trace log ...
Trace on (Clear trace log )
For lCount from 1 to 5 step 1
  OK message {Sent to trace log}
End For
Trace off
# ...and the following line is not
OK message {Not sent to trace log}
```

Transmit text to port

Command group	Flag affected	Reversible
Reports and Printing	YES	NO

Syntax

Transmit text to port ([*Add newline*]) {*text*}

Options

Add newline	If specified, the command sends a newline character sequence after sending the text

Description

This command sends text to a port; for example, you can send printer control characters. To transmit control characters, you can use the *chr()* function inside square brackets. For example, [*chr(27,14)*] sends escape 14.

The *Add newline* option enables you to send end of line characters after each line of text.

An error occurs and the flag is cleared if the port has not been selected or if the user presses Ctrl-Break/Ctrl-C/Cmnd-period while waiting for the output buffer to be emptied.

When you use a printer connected to the port, this command lets you send escape codes to control print characteristics.

Example

```
# Send text followed by the report rMyReport to a port
Set port name {1 (Modem port)}
Set port parameters {1200,n,7,2}
Transmit text to port (Add newline) {This is my report}
Set report name rMyReport
Send to port
Print report
Close port
```

Transmit text to print file

Command group	Flag affected	Reversible
Reports and Printing	YES	NO

Syntax

Transmit text to print file ([*Add newline*]) {*text*}

Options

Add newline	If specified, the command sends a newline character sequence after sending the text

Description

This command sends text to a print file, for example, you can send printer control characters. To transmit control characters, you can use the *chr()* function inside square brackets. For example, [*chr(27,14)]*sends escape 14.

The *Add newline* option causes Omnis to add end of line characters after each line of text.

An error occurs if no print file has been selected.

Example

```
# Create a file containing the text 'This is my report' together with the report rMyReport
Set print or export file name {[con(sys(115),'output.txt')]}
Transmit text to print file (Add newline) {This is my report}
Set report name rMyReport
Send to file
Print report
Close print or export file
```

Truncate file

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

Truncate file (*refnum* [,*end-position*] [,*end-position-is-character*]) **Returns** *err-code*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This command truncates a file. You specify the file reference number returned by Open file in *refnum*. The file is truncated at the current position of the file pointer or the *specified end-position* if given. The *end-position* parameter represents a byte position, unless you pass end-*position-is-character* as a non-zero value, in which case it represents an operating system character position (a byte position when running on Linux, or a 16-bit character position when running on Win32 or macOS).

It returns an error code (See Error Codes), or zero if no error occurs.

Example

```
# write the contents of the character variable 'lCharVar' to a text file named 'charfile.txt' in the root of t
Calculate lPathname as con(sys(115),'charfile.txt')
Create file (lPathname) Returns lErrCode
Open file (lPathname,lRefNum)
Calculate lCharVar as 'Truncate the contents of this file'
Write file as character (lRefNum,lCharVar) Returns lErrCode
Truncate file (lRefNum,8) Returns lErrCode
Close file (lRefNum)
```

Uncheck menu line

Command group	Flag affected	Reversible	Execute on client
Menus	NO	YES	NO

Syntax

Uncheck menu line *line* or *instance-name/line*

Description

This command removes the check mark on the specified line of a menu instance. No action is taken if there is no check mark or the menu instance is not installed. You specify the menu-instance-name and the line-number of the menu line you want to uncheck.

If you use **Uncheck menu line** in a reversible block, the specified menu line is checked again when the method terminates.

Example

```
# Test whether a line in the menu instance is checked and
# either check or uncheck it accordingly.
Install menu mView
Test for menu line checked mView/Large
If flag true
  Uncheck menu line mView/Large
Else
  Check menu line mView/Large
End If
# Alternatively, you change the $checked property of a line
# in the menu instance using notation
Do $imenus.mView.$objs.Large.$checked.$assign(kFalse)
```

Unload error handler

Command group	Flag affected	Reversible	Execut
Error handlers	YES	NO	NO

Syntax

Unload error handler [*name/*]*name*

Description

This command unloads the specified error handler (a method is taken as its parameter). If there are multiple error handlers at that method, they are all unloaded. The flag is set if an error handler is unloaded. See Load error handler for more information about error handlers.

Example

```
Unload error handler cMyErrorHandler/Errors
Load error handler cMyErrorHandler/Error2Handler
```

Unload event handler

Command group	Flag affected	Reversible	Execute on client
Externals	NO	NO	NO

Syntax

Unload event handler routine-name or library-name/routine-name (parameters)

Description

This command unloads the specified event handler or, if no handler is specified, all event handlers. If none exists, no action is taken. An event handler is always unloaded when the library is closed or when the program quits. See Load event handler for more information on event handlers.

Example

```
# unload the event handler, myEventHandler
Unload event handler myEventHandler
```

Unload external routine

Command group	Flag affected	Reversible	Execute on client
Externals	YES	NO	NO

Syntax

Unload external routine *routine-name* or *library-name/routine-name* (parameters)

Description

This command unloads the specified external code from memory. If it is not already loaded or is not found, the flag is cleared and no action takes place. If no external is specified, all externals are unloaded. All loaded external routines are unloaded when the library is closed or when the program quits. See Load external routine for more information on external routines.

Example

```
Unload external routine** MathsLib/sqroot
```

Until break

Command group	Flag affected	Reversible	Execute on client
Constructs	NO	NO	NO

Syntax

Until break

Description

This command terminates a repeat loop if the user requests a cancel by either clicking on a working message Cancel button, or by pressing Ctrl-Break under Windows, Ctrl-C under Linux, or Cmnd-period under macOS. Note that the user cannot request a cancel (and therefore cause **Until break** to terminate the repeat loop) if Disable cancel test at loops has been executed. Note that you can also terminate a repeat loop using Break to end of loop within the loop, or by using one of the alternative Until... commands.

Example

```
# only way out of this loop is to enter a value greater than 10
Disable cancel test at loops
Repeat
  Prompt for input Enter a value greater than 10 to exit loop Returns lValue
  If lValue>10
    Break to end of loop
  End If
Until break
```

Until calculation

Command group	Flag affected	Reversible	Execute on client
Constructs	NO	NO	YES

Syntax

Until *calculation*

Description

This command terminates a Repeat-Until conditional loop specifying a *calculation* as the condition. The *calculation* is evaluated at the end of the loop that continues if the derived value is zero.

Example

```
Calculate lCount as 1
Repeat ## Repeat loop
  Calculate lCount as lCount+1
Until lCount>=3
OK message {Count=[lCount]} ## prints 'Count=3'
```

Until flag false

Command group	Flag affected	Reversible	Execute on client
Constructs	NO	NO	YES

Syntax

Until flag false

Description

This command terminates the Repeat–Until conditional loop if the flag is false; execution continues with the command following the Until. If the flag is true, execution continues with the command following the Repeat.

Example

```
# loop until 'No' is pressed
Repeat
  No/Yes message {Press No to exit loop}
Until flag false
```

Until flag true

Command group	Flag affected	Reversible	Execute on client
Constructs	NO	NO	YES

Syntax

Until flag true

Description

This command terminates the Repeat–Until conditional loop if the flag is true; execution continues with the command following the Until command. If the flag is false, execution continues with the command following the Repeat command.

Example

```
# loop until 'Yes' is pressed
Repeat
  Yes/No message {Press Yes to exit loop}
Until flag true
```

Update data dictionary

Command group	Flag affected	Reversible	Ex
Data management	YES	NO	NO

Syntax

Update data dictionary ([*Test only*]) {*list-of-files* (F1,F2,..,Fn) (leave empty to select all)}

Options

Test only	If specified, the data file is not updated; the command purely tests to see if it would update the data file when executed without this option specified, and returns the flag set to true if an update would occur

Description

This command updates the data dictionary for the specified file or list of files. The data dictionary is a copy of the file class field definitions and is stored in the data file. The command lets you write minor file class changes to the data dictionary. These minor changes do not require data reorganization, and include changes such as adding new fields, altering field names and altering field lengths. You can only update the data dictionary if you are the only user logged on to the data file.

Update data dictionary updates the data dictionary for the specified list of file classes. If you omit a file name or list of files, all the files with slots in the current data file are updated.

If a specified file name does not include a data file name as part of the notation, the default data file for that file is assumed. If the file is closed or memory-only, the command does not execute and returns with flag false.

If the *Test only* option is specified, no updating is actually carried out, and the flag is set if at least one file in the data dictionary needs updating.

Certain changes made to a file class (that is, changes in indexes, field type changes and changes in file connections) require data reorganization. In this case, using **Update data dictionary** to keep the file class and the data file "in step" will be inappropriate. Reorganize data lets you test whether a data file needs reorganization as well as to reorganize it if necessary.

Example

```
Update data dictionary (Test only) ## all files
If flag true
  Update data dictionary ## all files
End If
```

Update files

Command group	Flag affected	Reversible	Execute c
Changing data	YES	NO	NO

Syntax

Update files ([*Do not cancel pfu*])

Options

Do not cancel pfu	If specified, Omnis remains in prepare for update mode after the command finishes, meaning that multi-user locks remain in place, and you can perform further updates

Description

This command writes the records in the current record buffer to disk and cancels the Prepare for... mode. You must execute the command when Omnis is in a Prepare for update mode otherwise an error occurs.

If a warning error code kerrUnqindex or kerrNonull is returned during the execution of this command, the Prepare for update mode is not canceled. This means that you can check for these errors and recover without losing the data the user has already typed in. In fact, if you issue a new Prepare for... command, Omnis will reread records, and any data that is already in the CRB will be lost.

The *Do not cancel pfu* option prevents the command from canceling Prepare for update mode. Thus, you can make more changes to the data, the multi-user locks remain in place, and another **Update files** can be executed.

The ***Update files*** command causes the indexes in the files to be re-sorted. Thus, in multi-user mode, the files are locked while Update files is executing. You can control this file locking by running Do not wait for semaphores. When Do not wait for semaphores is active, **Update files** returns flag false and does nothing if the file is locked.

Example

```
# The following example inserts an invoice in the parent file and a list of
# related invoice items in the child file. The Do not cancel pfu option ensures
#  that the parent record remains locked until complete.
Set main file {fInvoice}
Prepare for insert
Enter data
Update files (Do not cancel pfu)
Set main file {fItems}
For lInvoiceItems.$line from 1 to lInvoiceItems.$linecount step 1
  Prepare for insert
  Load from list
  Update files (Do not cancel pfu)
End For
Update files
# In multi-user mode you can control file locking using Do not wait for semaphores, for example
Wait for semaphores
Prepare for edit
Enter data
Do not wait for semaphores
If flag true
  Repeat
    Working message {Waiting for file locks}
    Update files
  Until flag true
End If
```

Update files if flag set

Command group	Flag affected	Reversible	Execute c
Changing data	YES	NO	NO

Syntax

Update files if flag set ([*Do not cancel pfu*])

Options

Do not cancel pfu	If specified, Omnis remains in prepare for update mode after the command finishes, meaning that multi-user locks remain in place, and you can perform further updates

Description

This command writes the current values in the current record buffer to disk if the flag is set to kTrue. This is a variation on the Update files command and is equivalent to:

```
If flag true
  Update files
End If
```

When the command follows Enter data, the Prepare for update mode is cancelled, and the record is stored on disk if the user clicks OK or presses the Return/Enter key.

UUDecode

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

UUDecode (*stream*,*decoded-stream*) **Returns** *status*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

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**.

Decoded-Stream 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. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# encode the contents of the character variable lString to get lEncodedString
# and decode lEnclodedString to get lString back
Calculate lString as 'This is my character string to encode'
UUEncode (lString,lEncodedString) Returns lErrCode
Calculate lString as ''
UUDecode (lEncodedString,lString) Returns lErrCode
```

UUEncode

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

UUEncode (*stream*,*encoded-stream*) **Returns** *status*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

UUEncode turns a stream into an encoded stream of ASCII characters. The encoded version is approximately 1.25 times larger than the original and can be decoded using UUDecode.

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

Encoded-Stream is an Omnis Character or Binary field that receives the resulting Uuencoded representation of the Stream parameter.

Status is an Omnis Long Integer field which receives the value zero for success, or an error code < 0 for failure. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# encode the contents of the character variable lString to get lEncodedString
Calculate lString as 'This is my character string to encode'
UUEncode (lString,lEncodedString) Returns lErrCode
```

Variable menu command

Command group	Flag affected	Reversible	Execute on client
Debugger	NO	NO	NO

Syntax

Variable menu command: *command* {*list-of-field-or-file-names* (F1,F2..F3,F4)}

Commands

Set Break On Variable Change
Clear Break On Variable Change
Set Break On Calculation
Clear Break On Calculation
Store Min & Max
Do Not Store Min & Max
Add To Watch Variables List
Remove From Watch Variables List
Send Value To Trace Log
Send Minimum To Trace Log
Send Maximum To Trace Log
Send All To Trace Log
Open Value Window
Open Values List...

Description

This command performs one of the Variable context menu options on the specified field or list of fields. You can specify one of the following Variable menu options:

Set break on variable change sets a variable change breakpoint for each variable in the list.

Clear break on variable change clears any variable change breakpoint for each variable in the list. If no variable names list is specified, all current variable change breakpoints are cleared.

Set break on calculation sets a calculation breakpoint for each variable in the list. You can set the calculation for each variable using Set break calculation. Setting calculation breaks for more than a very few variables will cause methods to run very slowly.

Clear break on calculation clears any variable change breakpoints for each variable in the list. If no variable names list is specified, all current calculation breakpoints are cleared.

Store min & max causes minimum and maximum values to be stored for each variable in the list.

Do not store min & max clears 'Store min and max' mode for each variable on the list. If no variables are specified, all current 'Store min and max' are cleared.

Add to watch variables list marks each variable on the list as a watch variable.

Remove from watch variables list marks each variable on the list as not watched. If no variables are specified, all variables are marked as not watched. Note that variables with breakpoints or with 'Store min and max' mode set always appear in the watch variables list.

Send value to trace log adds a line to the trace log for each variable on the list. If no variables are specified, all values for all variables on the watch variables list are sent to the trace log.

Send minimum to trace log adds a line to the trace log for each variable on the list for which 'Store min and max' is set. If no variables are specified, the minimum values for all variables for which 'Store min and max' is set are sent to the trace log.

Send maximum to trace log adds a line to the trace log for each variable on the list for which 'Store min and max' is set. If no variables are specified, the maximum values for all variables for which 'Store min and max' is set are sent to the trace log.

Send all to trace log adds a value line to the trace log for each variable on the list, and adds minimum and maximum line(s) to the trace log for each variable on the list for which 'Store min and max' is set. If no variables are specified, this is carried out for all appropriate variables on the watch variables list.

Open value window opens a value window for each variable on the list, or for every variable on the watch variables list if no variables are specified. There is a limit on the number of windows that you can open at once.

Open values list opens the values list for each of the variable types given in the command parameters. For example, ***Variable menu command:*** *open values list {lValue, iCount}* opens two values lists, one for Local variables, the other for Instancel variables. There is one values list for each file class, so if more than one variable name in a particular file class is specified the values list for that file will only be opened once. There is also a limit on the number of windows that you can open at once.

Example

```
# pause method execution when the value of iValue changes
Calculate iValue as 5
Variable menu command : Set Break On Variable Change {iValue}
Calculate iValue as 2
Variable menu command : Clear Break On Variable Change {iValue}
# open two variable windows one showing the value of lCount and the other the value of iValue
Calculate lCount as 10
Variable menu command : Open Value Window {lCount,iValue}
```

Wait for semaphores

Command group	Flag affected	Reversible	Execute c
Changing data	NO	YES	NO

Syntax

Wait for semaphores

Description

This command causes all the commands which set semaphores to wait with a lock cursor until the semaphores for the required records are available.

When a library is first selected, **Wait for semaphores** is automatically selected to ensure compatibility with existing libraries. It causes all the commands which set semaphores to wait with a lock cursor until the semaphore is available then return with the flag set, or to wait until the user cancels with a Ctrl-Break/Ctrl-C/Cmnd-period then return with a flag clear.

Semaphores

Semaphores are internal flags or indicators set in the data file to show other users that the record has been required elsewhere for editing. Semaphores are set only when running in multi-user mode, that is, the data file is located on a networked server, a Mac volume or on a DOS machine on which SHARE has been run.

The commands which set semaphores are Prepare for edit, Prepare for insert, Update files and Delete, and also, if pfu mode is on, Single file find, Load connected records, Next, Previous and Set read/write files. Auto finds on windows always wait for semaphores.

The Edit/Insert commands from the Commands menu always wait for a semaphore as do automatic find entry fields.

Example

```
Wait for semaphores
Prepare for edit ## waits for record if locked by another user
Enter data
Do not wait for semaphores
If flag true
  Update files
  If flag false
    OK message {File was locked, update failed}
  End If
End If
```

Web 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
-501	Incorrect parameter type
-502	Error getting information about a parameter
-503	Incorrect number of parameters
-504	The command can only decode streams of characters (in a character variable)
-506	Unrecognised command
-507	Error locking handle
-508	Bad list generated by command
-509	Bad socket passed to command
-511	No address specified
-512	Could not open ICMP handle
-513	Could not start timer
-516	The end-user cancelled the request
-517	Bad option passed to TCPClose
-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
-1012	Unix TCPPing requires a raw socket: only processes with an effective user id of zero or the CAP_NET_RAW capability are allowed to open raw sockets
-1013	Cannot do TCPPing on Win32 without icmp.dll
-1014	Attempt to perform secure operation on non-secure connection
-1015	Attempt to make an already secure connection secure
-1016	Cannot load wesecure.so/dll. Perhaps OpenSSL is not installed
-1017	A required function is missing from wesecure.so/dll
-1018	Cannot request a partial closure of a secure connection
-1019	Error setting up secure library threading
-1020	Error seeding Pseudo Random Number Generator
-1021	The OpenSSL library returned an error; call WebDevGetSecureError for more information
-1022	Invalid secure object passed to wesecure
-1023	Unknown error returned by OpenSSL library
-1024	Unable to get a suitable Omnis folder for CA certificates
-1025	Cannot open cacerts.pem
-1026	Cannot get find handle to list CA certificates
-1027	cacerts folder does not contain any CA certificates
-1028	Error getting next certificate file
-1029	Unable to open CA certificate
-1030	Wesecure initialization failed
-1031	Error establishing secure connection

Error Code	Error Text
-1032	Buffer overflow would occur because a field is too long
-1033	Connection gracefully closed
-1034	Attempt to connect timed out
-1035	Unknown error
-1036	InitSecurityInterface failed
-1037	Internal error with received data buffer
-1038	Internal error with extra data buffer
-1039	Could not find decryption output buffer
-1040	Unable to resume session for data connection
-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
-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)

Error Code	Error Text
-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
-1150	Must use passive FTP when the connection is secure
-1151	FTP server response to AUTH TLS command does not allow a secure connection to be established
-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
-1188	The maximum response size specified in the call to HTTPRead was exceeded
-1189	Proxy server rejected CONNECT method
-1190	Invalid HTTP authentication type
-1191	No HTTP method specified
-1192	The connection to the server has closed (this can occur for example due to the server timing out the connection)
-1202	SMTP: the server response to the STARTTLS command was incorrect
-1203	SMTP: the server does not support the STARTTLS command, so a secure connection cannot be established
-1204	SMTP: the secure parameter to SMTPSend is invalid
-1205	SMTP: 435 Unable to authenticate at present
-1206	The SMTP server does not support authentication
-1207	SMTP: 535 Incorrect authentication data
-1208	SMTP: 432 A password transition is needed
-1209	SMTP: 534 The authentication mechanism is too weak

Error Code	Error Text
-1210	SMTP: 538 Encryption is required for requested authentication mechanism
-1211	SMTP: 454 Temporary authentication failure
-1212	SMTP: 530 Authentication is required
-1213	Unexpected response from server during authentication
-1214	SMTP: OK Authenticated
-1215	SMTP: continue command
-1216	Required type of authentication (PLAIN or LOGIN) not supported by SMTP server
-1217	The response to the EHLO command could not be parsed
-1218	Parameter passed to mail command is too long
-1219	SMTP: 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
-1224	SMTP: 250 Requested mail action okay, completed
-1225	SMTP: 251 User not local; will forward to <forward-path>
-1226	SMTP: 354 Start mail input; end with <CRLF>.<CRLF>
-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 data
-1242	Body part list is inconsistent - cannot build MIME content

Error Code	Error Text
-1243	Header name is empty
-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)
-1247	POP3: the secure parameter is invalid
-1260	IMAP: invalid response received from server
-1261	IMAP: invalid tag received from server
-1262	IMAP: invalid greeting message received from server
-1263	IMAP: connection rejected by server (BYE received in greeting message)
-1264	IMAP: the server does not support plain or CRAM-MD5 authentication
-1265	IMAP: the server does not support the STARTTLS command, so a secure connection cannot be established
-1266	IMAP: the server response to the CAPABILITY command was incorrect
-1267	IMAP: the server must support IMAP4rev1, but it does not indicate that in its CAPABILITY response
-1268	IMAP: the server response to the STARTTLS command was incorrect
-1269	IMAP: parameter passed to command is too long
-1270	IMAP: login was rejected because the user name or password was incorrect
-1271	IMAP: the server response to the LOGIN command was incorrect
-1272	IMAP: the server response to the LOGOUT command was incorrect (network connection has been closed)
-1273	IMAP: the secure parameter to IMAPConnect is invalid
-1274	IMAP: the server response to the AUTHENTICATE CRAM-MD5 command was incorrect
-1275	IMAP: error decoding base64 response to AUTHENTICATE CRAM-MD5 command
-1276	IMAP: the server response to the LIST or LSUB command was incorrect
-1277	IMAP: the server response to the SELECT command was incorrect
-1278	IMAP: missing output list parameter
-1279	IMAP: the server response to the SUBSCRIBE command was incorrect
-1280	IMAP: the server response to the UNSUBSCRIBE command was incorrect
-1281	IMAP: the server response to the STORE command was incorrect
-1282	IMAP: the server response to the RENAME command was incorrect
-1283	IMAP: the server response to the EXPUNGE command was incorrect

Error Code	Error Text
-1284	IMAP: the server response to the DELETE command was incorrect
-1285	IMAP: the server response to the CREATE command was incorrect
-1286	IMAP: the server response to the COPY command was incorrect
-1287	IMAP: Incorrect FETCH response message sequence number when listing messages
-1288	IMAP: the server response to the FETCH command was incorrect when listing messages
-1289	IMAP: No flags returned in FETCH response when listing messages
-1290	IMAP: No size returned in FETCH response when listing messages
-1291	IMAP: No UID returned in FETCH response when listing messages
-1292	IMAP: Flags in response are not terminated by close parenthesis
-1293	IMAP: Invalid integer value in FETCH response when listing messages
-1294	IMAP: FETCH response not terminated with close parenthesis when listing messages
-1295	IMAP: No INTERNALDATE returned in FETCH response when listing messages
-1296	IMAP: INTERNALDATE returned in FETCH response when listing messages is not correctly enclosed in double quotes
-1297	IMAP: FETCH response incomplete
-1298	IMAP: the server response to the FETCH command was incorrect when receiving a message or headers
-1299	IMAP: bad message length in FETCH response
-1300	IMAP: unrecognized data item in response when FETCHing message or headers
-1301	IMAP: the server response to the NOOP command was incorrect
-1302	IMAP: the server response to the CHECK command was incorrect
-1303	IMAP: The length of the additional headers requested is too long
-1304	IMAP: FETCH response invalid
-1305	IMAP: Selected headers not returned in FETCH response when listing messages
-1306	IMAP: Invalid header line in FETCH response
-1307	IMAP: Received a header that was not requested in a FETCH response
-10004	Socket error: Interrupted function call
-10009	Socket error: Bad file descriptor
-10013	Socket error: Permission denied
-10014	Socket error: Bad address
-10022	Socket error: Invalid argument
-10024	Socket error: Too many open files
-10035	Socket error: The command would block
-10036	Socket error: Operation now in progress
-10037	Socket error: Operation already in progress

Error Code	Error Text
-10038	Socket error: Socket operation on non-socket
-10039	Socket error: Destination address required
-10040	Socket error: Message too long
-10041	Socket error: Protocol wrong type for socket
-10042	Socket error: Bad protocol option
-10043	Socket error: Protocol not supported
-10044	Socket error: Socket type not supported
-10045	Socket error: Operation not supported
-10046	Socket error: Protocol family not supported
-10047	Socket error: Address family not supported by protocol family
-10048	Socket error: Address already in use
-10049	Socket error: Cannot assign requested address
-10050	Socket error: Network is down
-10051	Socket error: Network is unreachable
-10052	Socket error: Network dropped connection on reset
-10053	Socket error: Software caused connection abort
-10054	Socket error: Connection reset by peer
-10055	Socket error: No buffer space available
-10056	Socket error: Socket is already connected
-10057	Socket error: Socket is not connected
-10058	Socket error: Cannot send after socket shutdown
-10059	Socket error: Too many references; cannot splice
-10060	Socket error: Connection timed out
-10061	Socket error: Connection refused
-10062	Socket error: Too many levels of symbolic links
-10063	Socket error: File name too long
-10064	Socket error: Host is down
-10065	Socket error: No route to host
-10066	Socket error: Directory not empty
-10067	Socket error: Too many processes
-10068	Socket error: Too many users
-10069	Socket error: Disk quota exceeded
-10070	Socket error: Stale NFS file handle
-10071	Socket error: Too many levels of remote in path
-10091	Socket error: Network subsystem is unavailable
-10092	Socket error: WINSOCK.DLL version out of range
-10093	Socket error: Successful WSAStartup() not yet performed
-10101	Socket error: Graceful shutdown in progress
-11001	Lookup error: Host not found
-11002	Lookup error: Non-authoritative host not found
-11003	Lookup error: This is a non-recoverable error

Error Code	Error Text
-11004	Lookup error: Valid name, no data record of requested type

WebDevGetSecureError

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

WebDevGetSecureError () **Returns** *errortext*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

If you are using, or attempting to use, a secure connection to a server, the installed SSL library (that provides the ability to establish secure connections) may return useful error text. When this occurs, the Web command returns the error code -1021. You can use **WebDevGetSecureError** to return this error text.

Omnis maintains a separate copy of the secure error text, for each thread in the multi-threaded server.

Example

```
Calculate lServer as 'my.pop3.server'
Calculate lUserName as 'myusername'
Calculate lPassword as 'mypassword'
POP3Connect (lServer,lUserName,lPassword,"",kTrue) Returns iSocket
If iSocket=-1021
  WebDevGetSecureError Returns lSecureErrorText
End If
```

WebDevSetConfig

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

WebDevSetConfig ([*errorproc*,*commstimeout*]) **Returns** *status*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This Web command is multi-threaded, allowing another thread to execute in the multi-threaded server while it runs. Note that the same socket cannot safely be used concurrently by more than one thread.

WebDevSetConfig allows you to set some configuration options for the WEB commands. The WEB commands are the commands with names prefixed by CGI, FTP, HTTP, POP3, TCP, and UU, and the MailSplit and SMTPSend 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.
- If an error occurs, and there is no WebDevError method, return the error code.
- If an error occurs, and there is a WebDevError method, call the WebDevError method, and then 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:

- A character parameter containing an error message.
- A long integer containing the error code.
- 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 connect, or 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 1/60th second ticks, and stores the resulting unsigned long integer. If you pass 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. A negative value will also cause the command to ignore server Blocking errors; in this case, the timeout will be the negated value of the parameter.

The **WebDevSetConfig** command returns a long integer Status. Zero for success, or less than zero if an error occurs. Possible error codes are listed in the Web Command Error Codes Appendix.

Example

```
# call method $error in the current window instance a web error occurs

# $construct
WebDevSetConfig (con($cinst().$name,'.$error'))
# $error method
OK message Error executing [pCommand] {[pErrorCode] : [pErrorMsg]}

# $event of push button - force error so $error gets called
FTPConnect ('ftp.unknownserver.net','Username','Password') Returns lFTPSocket
```

While calculation

Command group	Flag affected	Reversible	Execute on client
Constructs	NO	NO	YES

Syntax

While *calculation*

Description

This command starts a While–End While loop that continues while a calculated condition remains true. When the condition is not satisfied the method jumps out of the loop and the first command after the closing End While is executed. A loop that begins with a While command must terminate with an End While otherwise an error occurs.

Example

```
Calculate lCount as 1
While lCount<=3 ## While loop
  Calculate lCount as lCount+1
End While
OK message {Count=[lCount]} ## prints 'Count=4'
```

While flag false

Command group	Flag affected	Reversible	Execute on client
Constructs	NO	NO	YES

Syntax

While flag false

Description

This command starts a While–End While loop that continues while the flag is false. While the condition is false, a command or a series of commands is executed until the condition becomes true, at which time the first command after the closing End While is executed. A loop that begins with a While command must terminate with an End While, otherwise an error occurs.

Example

```
# loop until 'Yes' is pressed
Calculate #F as kFalse
While flag false
  No/Yes message {Do you wish to stop looping}
```

While flag true

Command group	Flag affected	Reversible	Execute on client
Constructs	NO	NO	YES

Syntax

While flag true

Description

This command starts a While–End While loop which continues while the flag is true. While the condition is true, a command or a series of commands is executed until the condition becomes false, at which time the first command after the closing End While command is executed. A loop that begins with a While command must terminate with an End While, otherwise an error occurs.

Example

```
# loop until 'No' is pressed
Calculate #F as kTrue
While flag true
  Yes/No message {Do you wish to continue looping ?}
End While
```

Working message

Command group	Flag affected	Reversible	Execute c
Message boxes	NO	NO	NO

Syntax

Working message *title* ([*Cancel button*] [,*Repeat count*] [,*Do not auto close*]) {*message*}

Options

Cancel button	If specified, the message has a cancel button
Repeat count	If specified, the message displays a numeric repeat count (an internal value that increments during method execution)
Do not auto close	If specified, open a modal working message that does not automatically close when the method ends. While the message is open, subsequent working message calls with this option increment the message and ignore the other parameters

Description

This command displays a window, usually to indicate that Omnis is working or waiting for input. The window displays a sequence of changing icons to indicate that Omnis is actively working. A working message automatically closes when the method quits, and control returns to the user.

The *title* parameter contains a title for the working message window, together with parameters that configure the behavior and appearance of the window. The method editor has a Configure Working Message Helper button below the code entry field that you can use to easily set these parameters. The *title* parameter has the following syntax:

```
&lt;*Title
text*&gt; / &lt;*size*(16|32|48)&gt; : &lt;*color*&gt; : *id1,id2,...,idN* ; &lt;*speed*&gt; ; &lt;*progress
bar range*&gt;;&lt;*display delay*&gt;
```

The *size* specifies the width and height of the icons, and can be either 16, 32 or 48.

The *color* specifies the color to be applied to each icon, when the icon is a themed SVG icon. To specify a color, use either the name of an Omnis color constant e.g. kRed, or a 7 character hex encoding of the color, in the form #RRGGBB e.g. #0000FF is blue.

id1,id2,...,idN is a comma separated list of icon ids, that specifies the sequence of icons to be displayed in the working message window. These icons can come from an icon set accessible to the library containing themed SVG icons (or for legacy apps #ICONS for the library, or an icon data file such as OmnisPic).

speed indicates the time, in 1/60th second units, for which an icon in the sequence is displayed.

progress bar range specifies the range of a progress bar. If you specify a non-zero value, then the working message window displays a progress bar, and each call to the Working message command increases the length of the bar until it reaches the range.

display delay specifies the time in 1/60th second units, that must elapse before the working message window becomes visible. This allows you to use the Working message command in situations where the processing is sometimes very rapid, and in that case avoid the message displaying and disappearing almost immediately.

If a working message is placed in a loop with a *Cancel button*, pressing Ctrl-break/Ctrl-C/Cmnd-period or clicking on Cancel quits all methods. However, if you first execute Disable cancel test at loops, you can implement an orderly exit. If Disable cancel test at loops is executed before the loop, the cancel is detected only on executing the Working message.

A *Repeat count* option is available with **Working message**, and displays the value of an internal counter which indicates the number of times a particular **Working message** has been encountered. If the command is in a Repeat loop, the counter increments at each pass of the loop.

Example

```
# Working message with orderly exit
Begin reversible block
  Disable cancel test at loops
End reversible block
Working message (Cancel button) {Processing Record [lCount]}
For lCount from 1 to 20000 step 1
  Redraw working message
  If canceled
    Break to end of loop
  End If
End For
OK message {All done}
```

Write entire file

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

Write entire file (*path*, *binary-variable*) **Returns** *err-code*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This command writes an entire file from a binary field, previously populated by using Read entire file. It returns an error code (See Error Codes), or zero if no error occurs. The *Binary value* is in the following format:

- 12 byte header containing the Type (4 bytes), Creator (4 bytes), and File length (4 bytes).
- File data.

Example

```
# read and then write the binary contents of mypicture.jpg
# sys(115) returns the full path to the Omnis executable
Calculate lPathname as con(sys(115),'mypicture.png')
Read entire file (lPathname,lBinfld) Returns lErrCode
Write entire file (lPathname,lBinfld) Returns lErrCode
```

Write file as binary

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

Write file as binary (*refnum*, *binary-variable* [,*start-position*]) **Returns** *err-code*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This command writes the contents of the specified *binary-variable* to a file. You specify the file reference number returned by Open file in *refnum*.

If you specify the *start-position*, writing begins at that byte (0 is the first byte in the file, 1 is the second byte, and so on), otherwise it begins at the current position (the first byte when the file is first opened).

It returns an error code (See Error Codes), or zero if no error occurs.

Example

```
# write the binary class data of the window class 'MyWindow' to a file named
# 'binfile' in the root of the omnis tree, path returned in sys(115)
Calculate lPathname as con(sys(115),'binfile')
Create file (lPathname) Returns lErrCode
Open file (lPathname,lRefNum)
Calculate lBinfld as $clib.$windows.MyWindow.$classdata
Write file as binary (lRefNum,lBinfld) Returns lErrCode
Close file (lRefNum)
```

Write file as character

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

Write file as character (*refnum*, *character-variable* [,*start-position*]) **Returns** *err-code*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

This command writes the contents of the specified *character-variable* to a file. You specify the file reference number returned by Open file in *refnum*.

If you specify the *start-position*, writing begins at that absolute character position (0 is the first character in the file, 1 is the second character, and so on), otherwise it begins at the current position (the first character when the file is first opened).

It returns an error code (See Error Codes), or zero if no error occurs.

Example

```
# write the contents of the character variable 'lCharVar' to a text file
# named 'charfile.txt' in the root of the omnis tree
Calculate lPathname as con(sys(115),'charfile.txt')
Create file (lPathname) Returns lErrCode
Open file (lPathname,lRefNum)
```

```
Calculate lCharVar as 'The contents of this file was written using the command Write file as character'
Write file as character (lRefNum,lCharVar) Returns lErrCode
Close file (lRefNum)
```

WriteBinFile

Command group	Flag affected	Reversible
External commands	YES	NO

Syntax

WriteBinFile (*pathname*, *binfld* [, *start* [, *length*]]) **Returns** *return-value*

Description

Note: The flag is set according to whether Omnis was able to make a call to this external command.

WriteBinFile writes binary data to the file system or data fork (not the resource fork).

Note for macOS Users: ReadBinFile and **WriteBinFile** are useful for reading and writing documents but not system and application files.

Pathname is a character field containing the full path of the file to which to write. If the output file does not already exist, **WriteBinFile** creates it.

Binfld is a binary field from which to write the data.

Start is an integer field specifying the byte position in the file where writing should begin. If the parameter is not used, the command defaults to 0 (zero), that is, the beginning of the file. To append data to an existing file, set Start to -1 (minus one).

Length is an integer field containing the number of bytes to write. If the parameter is not used, the value defaults to the length of the Binary field.

Return-value is an integer field that is the number of bytes written if no error code is returned. Otherwise, it is an error code, one of:

-1: End of file

-2: Out of memory

-10: File not found

-11: Bad file name

-12: Volume not found

-20: IO error

-100: Incorrect number of parameters

-101: Bad parameter value

-998: File too large

Example

```
# write the binary class data of the window class 'MyWindow' to a file named
# 'binfile' in the root of the omnis tree sys(115) returns the full path to
# the Omnis executable
Calculate lPathname as con(sys(115),'binfile')
Calculate lBinfld as $clib.$windows.MyWindow.$classdata
WriteBinFile (lPathname,lBinfld) Returns lNumbytes
OK message {[lNumbytes] bytes written}
```

XOR selected and saved

Command group	Flag affected	Reversible	Execute on client
List lines	YES	NO	NO

Syntax

XOR selected and saved ([*All lines*]) {*line-number* (*calculation*)}

Options

All lines	If specified, the command affects all the lines in the list

Deprecated Command

This command has been deprecated and is no longer visible in the Code Assistant in the Code Editor (it will not appear when you type the first few characters), although it is still present in Omnis Studio and will continue to function if used in legacy code. You can show this command by disabling the appropriate Command Filter in the **Modify** menu in the Code Editor.

Description

This command performs a logical XOR of the Saved selection with the Current selection. To allow sophisticated manipulation of data via lists, a list can store two selection states for each line; the "Current" and the "Saved" selection. The Current and Saved selections have nothing to do with saving data on the disk; they are no more than labels for two sets of selections. The lists may be held in memory and never saved to disk: they will still have a Current and Saved selection state for each line but they will be lost if not saved. When a list is stored in the data file, both sets of selections are stored.

You can specify a particular line in the list by entering either a number or a calculation.

The **XOR selected and saved** command performs a logical XOR (exclusive OR) on the Saved and Current state and puts the result into the Current selection. Hence, if either of the Current and Saved states is selected, the Current state becomes selected, but if both states are equal, the resulting Current state will become deselected.

Logic Table (S=selected, D=deselected)

Saved	Current	Resulting Current State
S	S	D
D	S	S
S	D	S
D	D	D

The *All lines* option performs the XOR for all lines of the current list. The flag is set by this command. The following example selects the middle line of the list:

Example

```
# Leave line 3 selected
Set current list lMyList
Define list {lCol1}
For lCol1 from 1 to 6 step 1
  Add line to list {lCol1}
End For
Select list line(s) (All lines)
Save selection for line(s) (All lines)
Invert selection for line(s) {3}
XOR selected and saved (All lines)
```

Yes/No message

Command group	Flag affected	Reversible	Execute c
Message boxes	YES	NO	NO

Syntax

Yes/No message *title* ([*Icon*][,*Sound bell*][,*Cancel button*]) {*message*}

Options

Icon	If specified, the message displays an operating system specific icon
Sound bell	If specified, the system bell sounds when the command displays the message
Cancel button	If specified, the message has a cancel button

Description

This command displays a message box containing the specified message and provides a Yes and a No pushbutton. You can include a *Cancel button*, and add a short *title* for the message box. For greater emphasis, you can select an *Icon* for the message box (the default "info" icon for the current operating system), and you can force the system bell to sound by checking the *Sound bell* check box. Under Windows XP, you have to specify a system sound for a 'Question' in the Control Panel for the Sound Bell option to work.

When the message box is displayed method execution is halted temporarily; it remains open until the user clicks on one of the buttons before continuing. The Yes button is the default button and can therefore be selected by pressing the Return key.

The number of lines displayed in the message box depends on your operating system, fonts and screen size. In the message text you can force a break between lines (a carriage return) by using the notation '//'or the *kCr* constant enclosed in square brackets, e.g. 'First line[kCr]Second line'.

You can insert a **Yes/No message** at any appropriate point in a method. If the user clicks the Yes button, the flag is set; otherwise, it is cleared. You can use the msgcancelled() function to detect if the user pressed the Cancel button.

Example

```
# Open a Yes/No dialog and display the option selected
Yes/No message My Editor (Icon,Cancel button) {Do you wish to save the changes you have made ?}
If msgcancelled()
  OK message My Editor {Cancel button pressed}
Else
  If flag true
    OK message My Editor {OK button pressed}
  Else
    OK message My Editor {Cancel button pressed}
  End If
End If
```

Yield to other threads

Command group	Flag affected	Reversible	Execute on client
Threads	NO	NO	NO

Syntax

Yield to other threads

Description

Yield to other threads is only applicable to the multithreaded server.

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

Example

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

Calculations

Commands

Calculate	Do	Do default	Do inherited
Do redirect	JavaScript:	Set reference	Test for valid calculation

Changing data

Commands

The Changing Data commands are no longer visible in the Code Assistant in the Code Editor (they will not appear when you type the first few characters), although they are still present in Studio 11 and will continue to function. You can show these commands by disabling the appropriate Command Filter in the Modify menu in the Code Editor.

These commands are for desktop apps only, not web or mobile apps.

Cancel prepare for update	Delete	Delete with confirmation	Do not flush data
Do not wait for semaphores	Flush data	Flush data now	Prepare for edit
Prepare for insert	Prepare for insert with current values	Test for only one user	Update files
Update files if flag set	Wait for semaphores		

Classes

Commands

These commands are for desktop apps only, not web or mobile apps.

Close all designs	Close design	Delete class	Duplicate class
Modify class	Modify methods	New class	Print class
Rename class	Revert class	Save class	Set class description

Clipboard

Commands

These commands are for desktop apps only, not web or mobile apps.

Clear data	Copy to clipboard	Cut to clipboard	Paste from clipboard
Test clipboard			

Constructs

Commands

# Comment	Begin reversible block	Break to end of loop	Break to end of switch
Case	Default	Disable cancel test at loops *	Else
Else If calculation	Else If flag false	Else If flag true	Enable cancel test at loops *
End For	End If	End reversible block	End Switch
End While	For each line in list	For field value	If calculation
If canceled	If flag false	If flag true	Jump to start of loop
Repeat	Switch	Until break	Until calculation
Until flag false	Until flag true	While calculation	While flag false
While flag true			

**These commands are for desktop apps only, not web or mobile apps.*

Data files

Commands

The Data File commands are no longer visible in the Code Assistant in the Code Editor (they will not appear when you type the first few characters), although they are still present in Studio 11 and will continue to function. You can show these commands by disabling the appropriate Command Filter in the Modify menu in the Code Editor.

These commands are for desktop apps using Omnis data files only, not web or mobile apps.

Close data file	Close lookup file	Create data file	Floating default data file
Open data file	Open lookup file	Prompt for data file	Set current data file
Set default data file			

Data management

Commands

The Data Management Commands are no longer visible in the Code Assistant in the Code Editor (they will not appear when you type the first few characters), although they are still present in Studio 11 and will continue to function. You can show these commands by disabling the appropriate Command Filter in the Modify menu in the Code Editor.

These commands are for desktop apps using Omnis data files only, not web or mobile apps.

Build indexes	Check data
Delete data	Drop indexes
Print check data log	Quick check
Test check data log	Update data dictionary

Debugger

Commands

Breakpoint	Clear trace log	Close trace log	Open trace log
Print trace log	Send to trace log	Set break calculation	Trace off
Trace on	Variable menu command		

Enter data

Commands

These commands are for desktop apps only, not web or mobile apps.

Disable enter & escape keys	Enable enter & escape keys	Enter data

Error handlers

Commands

Load error handler	*SEA continue execution*	*SEA repeat command*	*SEA report fatal error*
Signal error	*Unload error handler*		

Events

Commands

These commands are for handling events in web or mobile apps.

On	On default

These commands are for desktop apps only, not web or mobile apps.

Process event and continue	Queue bring to top	Queue cancel	Queue click
Queue close	Queue double-click	Queue key-board event	Queue OK
Queue quit	Queue scroll	Queue set current field	Queue tab
Quit event handler			

Exchanging data

Commands

The Exchanging Data commands are no longer visible in the Code Assistant in the Code Editor (they will not appear when you type the first few characters), although they are still present in Studio 11 and will continue to function. You can show these commands by disabling the appropriate Command Filter in the Modify menu in the Code Editor.

These commands are for desktop apps only, not web or mobile apps.

Accept advise requests	*Accept commands*	*Accept field requests*	*Accept field values*
Advise on find/next/previous	*Advise on OK*	*Advise on redraw*	*Cancel advises*
Clear DDE channel item names	*Close DDE channel*	*Message timeout*	*Open DDE channel*

Request advises	*Request field*	*Send advises now*	*Send com-mand*
Send field	*Set advise options*	*Set DDE channel item name*	*Set DDE channel number*
Set server mode			

External commands

The HTTP, IMAP, POP3, FTP & some File Commands in the OWEB external have been deprecated in Studio 11, and you should use the equivalent methods in the OW3 Worker Object (if available).

These commands are no longer visible in the Code Assistant in the Code Editor (they will not appear when you type the first few characters), although they are still present in Studio 11 and will continue to function. You can show the External Commands by disabling the appropriate Command Filter in the Modify menu in the Code Editor.

Calling External Code

Call DLL	Register DLL

Email Commands

IMAPCheck	IMAPConnect	IMAPCopyMessage	IMAPCreateMailbox
IMAPDeleteMailbox	IMAPDisconnect	IMAPExpunge	IMAPListMailboxes
IMAPListMessages	IMAPListSubscribedMailboxes	IMAPNoOp	IMAPRecvHeaders
IMAPRecvMessage	IMAPRenameMailbox	IMAPSelectMailbox	IMAPSetMessageFlags
IMAPSubscribeMailbox	IMAPUnsubscribeMailbox	MailSaveMime	POP3Connect
POP3DeleteMessage	POP3Disconnect	POP3ListMessages	POP3MessageCount
POP3Recv	POP3RecvHeaders	POP3RecvMessage	POP3Stat
POP3UndoDeletes	SMTPSend		

FTP Commands

FTPChmod	FTPConnect	FTPCwd	FTPDelete
FTPDisconnect	FTPGet	FTPGetBinary	FTPGetLastStatus
FTPList	FTPMkdir	FTPPut	FTPPutBinary
FTPPwd	FTPReceiveCommandReply	FTPRemoveDir	FTPSendCommand
FTPSetConfig	FTPSite	FTPType	

TCP Commands

TCPAccept	TCPAddr2Name	TCPBind	TCPBlock
TCPClose	TCPConnect	TCPGetMyAddr	TCPGetMyPort
TCPGetRemoteAddr	TCPListen	TCPName2Addr	TCPPing
TCPReceive	TCPSend	TCPSocket	

Web Commands

CGIDecode	CGIEncode	HTTPClose	HTTPGet
HTTPHeader	HTTPMethod	HTTPOpen	HTTPPage
HTTPParse	HTTPPost	HTTPRead	HTTPSend
HTTPServer	HTTPSetAuthentication	HTTPSetProxyServer	HTTPSplitHTML
HTTPSplitURL	UUDecode	UUEncode	WebDevGetSecureError
WebDevSetConfig			

File Commands

Change working directory	Close file	Copy file	Create directory
Create file	Delete file	Does file exist	Get file info
Get file name	Get file read-only attribute	Get files	Get folders
Get working directory	Move file	Open file	Put file name
Read entire file	Read file as binary	Read file as character	ReadBinFile
Set file read-only attribute	Split path name	Truncate file	Write entire file
Write file as binary	Write file as character	WriteBinFile	

Externals

Commands

Build externals list	*Call external routine*	*Load event handler*	*Load external routine*
Unload event handler	*Unload external routine*		

Fields

Commands

These commands are for desktop apps only, not web or mobile apps.

Disable fields	*Enable fields*	*Hide fields*	*Redraw*
Redraw lists	*Show fields*	*Test for field enabled*	*Test for field visible*

Files

Commands

These commands are for desktop apps using Omnis data files only, not web or mobile apps.

Build field names list	Build file list	Clear all files	Clear main & connected
Clear main file	Clear range of fields	Clear selected files	Set closed files
Set main file	Set memory-only files	Set read-only files	Set read/write files

Finding data

Commands

The Finding Data commands are no longer visible in the Code Assistant in the Code Editor (they will not appear when you type the first few characters), although they are still present in Studio 11 and will continue to function. You can show these commands by disabling the appropriate Command Filter in the Modify menu in the Code Editor.

These commands are for desktop apps using Omnis data files only, not web or mobile apps.

Clear find table	Disable relational finds	Enable relational finds	Find
Find first	Find last	Load connected records	Next
Previous	Prompted find	Single file find	Test for a current record

Test for a unique index value

Importing and Exporting

Commands

The Importing and Exporting commands are no longer visible in the Code Assistant in the Code Editor (they will not appear when you type the first few characters), although they are still present in Studio 11 and will continue to function. You can show these commands by disabling the appropriate Command Filter in the Modify menu in the Code Editor.

These commands are for desktop apps using Omnis data files only, not web or mobile apps.

Build export format list	Close import file	Enclose exported text in quotes	End export
End import	Export data	Export fields	Import data
Import field from file	Import field from port	Import fields	Prepare for export to file
Prepare for export to port	Prepare for import from client	Prepare for import from file	Prepare for import from port
Prompt for import file	Set import file name		

Libraries

Commands

Change user pass-word *	Close library	Create library	Open library
Prompt for library *			

**These commands are for desktop apps only, not web or mobile apps.*