OMNIS Studio Conversion

OMNIS Software

May 1997

Table of Contents

About This Manual

This manual introduces the new features in OMNIS Studio from an existing OMNIS user's point of view and describes how you convert your old OMNIS applications to OMNIS Studio. After reading this manual and converting your application you may like to read the *Using OMNIS Studio* manual.

Before you start, you should note that in OMNIS Studio some existing objects have been renamed to comply with object-oriented terminology or standard industry usage. These are described in the following chapters, but in particular

- formats are renamed *classes*
- procedures are renamed *methods*, and attributes are further divided into *properties* and *methods*
- and under MacOS only, you should Ctrl-click and not Option-click to open variable and context menus

Note that all references to "V3" in this manual refer to all versions of OMNIS 7 Version 3, including 3.0, 3.2, and 3.5.

What's New?

The first chapter in this manual introduces the new *object-oriented* features in OMNIS Studio, the new *classes* and objects, and the new *GUI* and design tools available in OMNIS Studio.

What's Changed?

The second chapter provides a summary of all the changes in OMNIS Studio, and lists all the objects that have been *renamed*, *removed*, or made *obsolete*, including functions, attributes, and commands.

Conversion and Compatibility

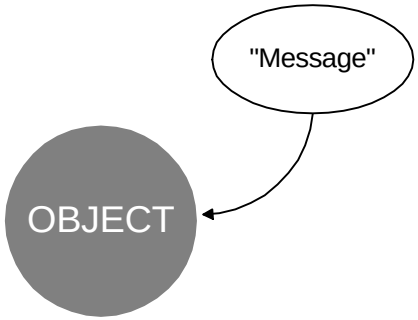
The final chapter in this manual describes how you *convert* your old OMNIS 7 Version 3 applications to OMNIS Studio and discusses the compatibility of converted libraries. If you want to convert an application created with an earlier version of OMNIS, such as OMNIS 7 Version 1 or 2 or even earlier, you must convert it to OMNIS 7 Version 3 first and use the information in this manual to convert it to OMNIS Studio. A demo version of OMNIS 7 Version 3 is provided for this purpose.

Chapter 1—What's New?

This chapter introduces the new features in OMNIS Studio, for existing OMNIS users and anyone who wants a quick summary of what's new in OMNIS. It includes a brief summary of the new object-oriented features in OMNIS Studio, the new classes and objects, and the new GUI and design tools in OMNIS Studio. For full details about all these topics, refer to the *Using OMNIS Studio* manual.

Object Orientation in OMNIS Studio

OMNIS Studio uses many of the concepts and terms in Object-Oriented Programming. The essential components of any OOP system are the *objects* in that system and the *messages* sent to and from those objects. OMNIS Studio is no exception.



OMNIS Studio expands the range and type of objects you have available, it enhances the system of messaging between these objects using a wealth of new notation, it increases your control over the events in your application using the new event handling system, and overall OMNIS Studio contains many new and improved design tools that let you and your development team create and manage the objects in your applications.

Libraries and Classes

An OMNIS *library* is a complex object that can contain many different types of GUI and data objects including windows, menus, toolbars, and reports. The definition for each of these objects is stored in your library as a separate *class*, called formats in previous versions. OMNIS Studio includes several new types of class including table, schema, code, and object classes.

The *Browser* lets you create libraries and classes, examine existing libraries and copy objects from one library to another. The *Component Store* lets you create classes and many other library objects, including window and toolbar controls, report fields, and external components.

Properties and Methods

In OMNIS Studio attributes are renamed and separated into *properties* and *methods*. Every object has certain characteristics that define exactly how it looks and behaves, known as its *properties*. Properties of an object could include its name, type, color, size, or visibility; all objects including libraries and classes have properties. Most objects also contain *methods*, which are pieces of code that perform some action when you send the object the appropriate message. For example, most GUI classes contain an $open() method that opens an instance of the class when the method is called.

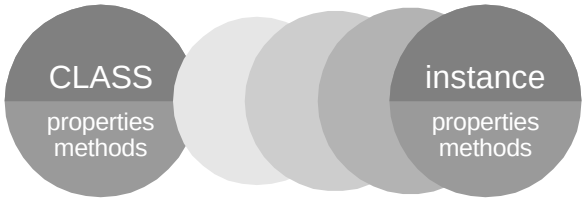
You can manipulate an object's properties and methods using the OMNIS notation. In the notation all property and method names start with a "$", and methods are further distinguished from properties by having parentheses after their name. You can examine and change the properties of an object using the *Property Manager*, and you can add methods to an object or change its default ones using the *Method Editor*. The code for a method can contain OMNIS commands, some notation, or a combination of the two.

Variables

The principal data container in OMNIS Studio is the *variable*. You add variables to an object using the *method editor*. Most objects can contain variables, but their scope and the kind of data they can contain depends on their type. OMNIS provides many different *data types* including character, number, date, list, and picture types. Furthermore, you can use the object class to create your own structured data objects containing their own encapsulated variables.

Instances

In OMNIS Studio, the principal objects that you manipulate at runtime are instances. An *instance* is the object you create when you *open* a class. For example, you create an instance of a window when you open a window class. When you print a report you create an instance of the report class, and similarly when you install a menu you create an instance of the menu class.

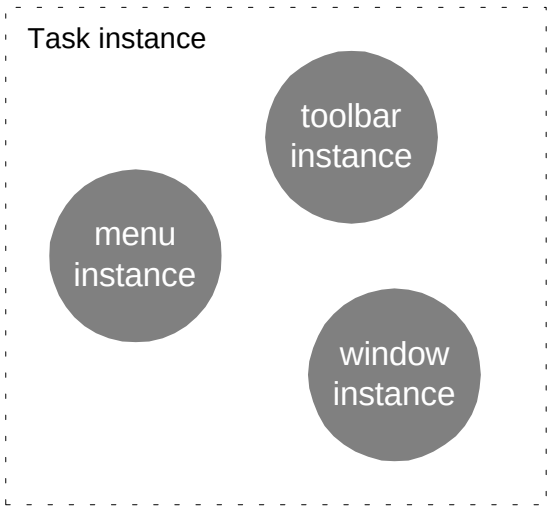


When you open an instance of a class it inherits all the properties, methods, and variables from the class. In addition to class variables, an instance can contain *instance variables*; these are defined in the class using the method editor. You can create multiple instances of most types of class, which means each instance can have its own set of instance variable *values*. For example, you can create multiple instances of the same window class and display different data values in each separate window instance.

Tasks

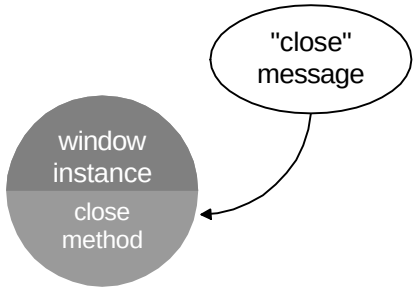
A *task* is a runtime container for the objects or instances in your application. You must open an instance of a class always from inside a task, that is, all instances must belong to a task. To ensure this, OMNIS opens a task for your instances to run in called the *startup task*, but you can create and open your own tasks.

The real advantage of using tasks is that you can control the instances belonging to a task simply by manipulating the task itself. For example, you can close all the instances belonging to a particular task by closing the task. OMNIS also hides and shows instances that belong to different tasks when you switch from one task to another.

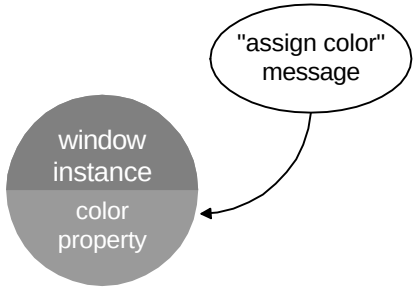


Messages

A *message* is an instruction to do something such as open, close, print, or redraw. You send messages to and from the various objects in your application using the *Do* command and the notation. A particular message will run the corresponding method contained in the object. Messages in OMNIS are formatted in the same way as the notation, where each message begins with a “$”, and then the name of the message. For example, you can send a $close() message to an open window which runs the standard *$close() method* in the window, which in turn closes the window instance.



You can activate or change object properties in a similar way using the $assign() message. When you assign to an object you can send a value or list of parameters with the message. For example, to change the background color of a window instance you would send the $backcolor.$assign() message with a color value as a parameter.

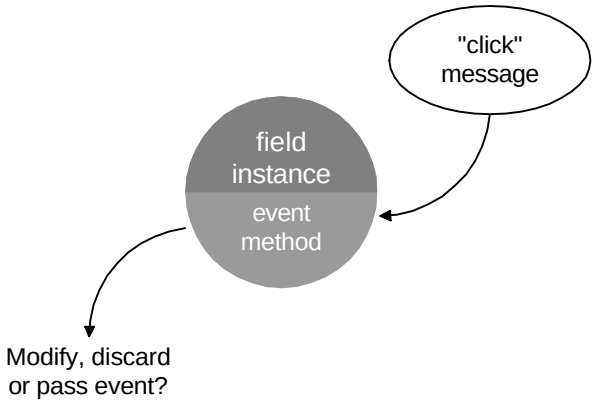


As well as assigning *to* an object, you can return information *from* an object using messages. For example, you could find out the background color of a window instance by returning the value of its color property.

Events

OMNIS Studio introduces a completely new event handling mechanism. Any user action that occurs in OMNIS Studio is reported as an *event message*. The key to creating an events-based application is in the methods you write to intercept or handle these events. These methods are called *event handling methods* and you would normally place them inside fields, windows, and tasks.

When the user generates an event, OMNIS sends a message describing the event. Usually this message is a simple predefined code for the event. Depending on where the event occurs it is intercepted by an event handling method. For example, if the user clicks on a field, a simple click event is generated and the event handler for the field is called.



The event handling method can modify the default action for the event or redirect method execution to anywhere else in the application. Alternatively, the event handler may discard the event altogether or pass it on to the next method in the event chain.

Inheritance

When creating a new class, instead of having to set up all its properties, methods, and objects you can derive it from an existing class. The new class is said to be a *subclass* of the existing class, which is in turn a *superclass*. In this way you can create a hierarchy of classes, each class being a subclass of the one above it in the chain. This saves you time and effort when developing your application, since you can reuse the objects from the superclass. When you make a change in a superclass, all its subclasses inherit the change automatically. You can view the inheritance structure for all the classes in your library using the *Inheritance Tree*.

New Classes and Objects

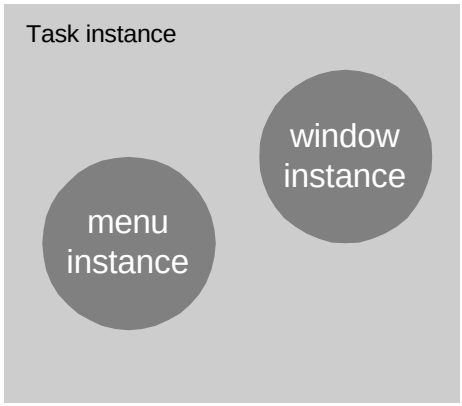
In OMNIS Studio formats are renamed *classes*. Some class types, such as file and search classes, are direct equivalents of the OMNIS 7 formats of the same name. Window classes have been greatly enhanced, and many new field types have been added. Other classes such as task, schema, and table classes are new in OMNIS Studio. This section summarizes the new classes in OMNIS and describes how the new object-oriented features are implemented.

Task Classes

A *task class* is a new type of class that lets you control or group instances. For example, you can link a menu instance with a particular window instance by opening them in the same task. Whenever the window comes to the top the menu instance is installed on the menu bar automatically.

You define exactly what happens in a task by creating methods in the task class. When you open a task class you create an instance of the task. A task instance can contain any other type of instance including window, report, and menu instances. When you open an instance from within a task instance it is said to *belong to* that task. By opening and closing different task instances, or by switching from one task to another you can control whole groups of instances.

When OMNIS starts up, a single *default task* is initiated, containing all the built-in design tools such as the Browser, the Component Store, and so on. When you create a new library it contains a task called the *startup task.* This task is run automatically when you open your library. From the startup task you can open other tasks containing other instances and objects that are part of your application.



When you close a task instance, all other instances belonging to that task are closed, providing they can be closed. When you quit OMNIS the default task is closed and all instances belonging to the default task are closed. Each library contains a group of task classes in $tasks, and all task instances are held in $itasks, in the order they are opened.

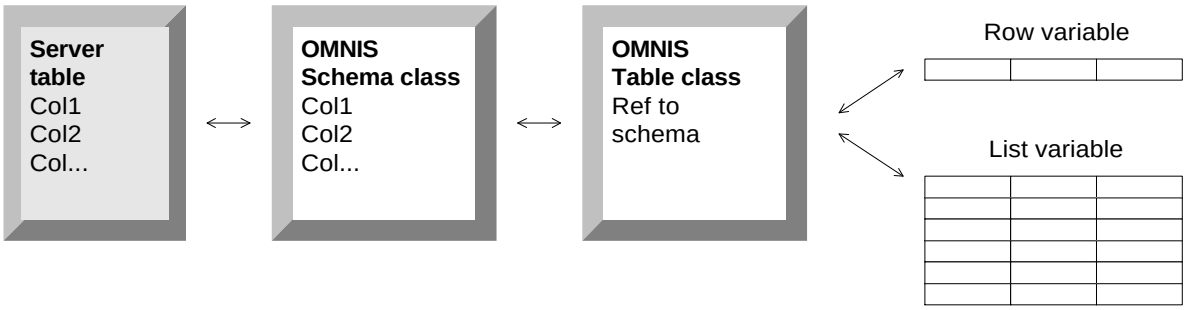
Schema and Table Classes

A *schema class* is a new type of class that represents a table or view in a server database. A schema class contains the name of the table or view on the server, a list of column names and data types for that table or view, and some additional indexing information about the columns.

The schema editor lets you enter the server table or view name, the column names, and data types for each column. Schema classes do not have methods and cannot be instantiated. Each library contains a $schemas group containing all the schema classes in your library.

A *table class* is a new type of class that provides an interface to the data modeled by a schema class. A table class contains the name of the schema class it uses, a number of standard methods for data handling, and your own custom properties and methods. Each library contains a $tables group containing all the table classes in your library.

A table class does *not* store data, rather you can use a table class to define a list or row variable which holds the data from your server database. You can define list and row variables using the *Define list from table* command, or the $definefromtable() method.



A list or row variable defined from a table class creates a table instance. Therefore, a list or row variable has its own properties and methods together with the properties and methods of a table instance. This includes methods you can use to fetch and insert data on your server database, such as $fetch() and $insert().

Construct and Destruct Methods

When you open a class, an instance of the class is created. Most classes that you can open or instantiate, including window, menu, toolbar, and report classes, contain two default methods called $construct() and $destruct(). The $construct() method controls the opening or construction of the instance. Therefore, you can place any code that you want to run as the class is opened in the $construct() method. Similarly, just before an instance is closed or destructed its $destruct() method is called. Any code that you place in the $destruct() method for a class is executed as the instance is closed or destructed. When you open a class, either using a command or the $open() method, you can send parameters to the $construct() method.

Fields can also contain a $construct() method. These are called in the order of the fields on the window, and before the class $construct() method. You can use this $construct() method to change the field as it is instantiated.

Window Classes

When you open a window class in OMNIS Studio you create an instance of the class. You can open a window class using the *Open window instance* command or the $open() method. You can open multiple instances of the same window class, providing they are all given unique names when you open them. Each window instance can have its own set of instance variable values which means you can display different sets of data in different windows at the same time. The $iwindows group contains all the current window instances, while the $windows group contains all the window classes in your library.

You can define class and instance variables for window classes. Window class variables are visible to the window class and all its instances, and instance variables are visible only to the instance they belong to.

You can add multiple methods to a window class, as well as the fields and objects in the window. The class methods for a window control the window as a whole, including the $construct() and $destruct() methods which manage the construction and destruction of the instance, and the $control() method which handles events in the window. Also you can use window class methods as custom properties and methods for the window.

Each window field can have multiple methods, including one called $event() used for handling events in the field. You can also use the field methods as custom properties and methods for the field. The field methods belong to the window class and share the same class and instance variables.

Window Fields

When you create a window class, the Component Store contains all the different field types and window objects available in OMNIS Studio. Many new types of window field have been added, including the following types.

List and Grid Fields

The OMNIS 7 V3 table field has been renamed the *complex grid* field and works in very much the same way as before. The standard list field has been renamed the *list box* field. In addition to these changes many other list and grid field types have been added. *string* and *data grids* let you display character and numeric data in spreadsheet form, *check lists* display a column of check boxes to indicate selected lines, *icon arrays* show columns and rows of icons in large or small view, and *tree lists* display hierarchical lists containing expandable nodes.

Tab and Page Fields

Tab panes and *page panes* contain a number of pages on which you can place other fields and controls. The *tab strip* field displays a number of tabs which the user can select and deselect.

Modify Report and Screen Report Fields

Modify report fields let you display a report class on an open window, which lets users change and move objects on the report. Screen report fields let you print a report to a window field, rather than the standard destinations such as the printer or to a file.

Group Fields

Shape fields, group boxes, and *scroll boxes* let you group other fields and objects on your window. Shape fields can also detect events including mouse clicks.

Report Classes

In OMNIS Studio when you print a report class, an instance of the class is created. The report instance exists for as long as it takes to send the report to the printer, or for a screen report until you close it. You can create any number of report instances, either from the same report class or any other report class.

You can use the $open() method or the *Prepare for Print* command to open a report instance, and you can terminate a report using the $close() method or the *End Print* command. The $ireports group contains all the report instances that are currently being processed. The most recently created report instance is at the top of the $ireports group.

You can send a list of parameters to the $construct() method when you open the report instance using the $open() method. You can send data to a report instance using the *Print report* or *Print record* commands. The *Print* commands have an option *Do not finish other*

reports which lets you print multiple report instances, one after another. Without this option any reports already in progress are terminated.

You can specify a report instance name for the *Print record* and *End print* commands. The default is to print or end the most recently started report.

Message Driven Reports

Having opened a report instance with the $open() method you can print your data using the *Print report* command, or for finer control, you can use a series of *Print record* commands followed by an *End print* command. Alternatively, you can send print messages to a report instance using the notation. For example, you can send a $printrecord() message to print a record to the report instance, or send an $endprint() message to finish the report. You can override the default handling for these messages by writing your own custom methods with the same name. You enter these custom methods in the class methods for the report class.

Every report field and section can contain a $print() method. When $print() is called for a particular section it calls the $print() method for all the fields in that section. You can send a parameter with the $print() message to override the default positioning of the section. You can write your own custom $print() methods for each field and section on a report. You enter these methods in the field methods for a report class. For any custom $print() method you can include the *Do default* command to run the default processing.

Subwindows

A *subwindow* is a new type of window field in which you can display another window class. The window class displayed in a subwindow field can contain one or more fields. For example, it could contain a set of general-purpose buttons and methods that you could reuse throughout your application as a subwindow on any number of other windows.

In design mode a subwindow field appears as a single field which you can move and resize like any other field. In runtime, all the fields and background objects in the subwindow become active fields on the parent window, so the user is not aware that they are from another class.

Regarding events, a subwindow is treated as a window within a window. Events that occur in the subwindow are handled by the subwindow event handling methods and then by the parent window event handling methods.

Field Styles

In OMNIS Studio you can format fields and text objects using styles. A *field style* is a style definition, like a word-processing or DTP style, that you can apply to window and report objects. Each field style contains a definition of its name, font, size, type style, and alignment. You can specify different combinations of text properties for each platform

under the same style name, so objects will display in the appropriate font and point size depending on the current operating systems.

OMNIS Studio provides some default styles for standard entry fields, pushbuttons, and lists, but you can add your own styles. The style for a particular window or report object is stored in its **fieldstyle** property. You can create as many field styles as you like and store them in the #STYLES system table. Having set up the styles in the style table, you can copy the table to any library and use its styles throughout your whole application.

Menu Classes

The user interface elements of a menu and its methods have been separated in OMNIS Studio. There is a separate menu editor in which you create the lines for your menu, and you edit the methods for each menu line in the method editor. You can install a menu instance on the main menu bar, create a hierarchical menu, a popup menu, or create your own context menus using a menu class.

When you install or open a standard menu, popup, or context menu, using the $open() method or an *Install* command, an instance of the menu class is created. You can install or open any number of menus, and create multiple instances of the same menu class. You can also add menus to the menubar of a window class.

The $imenus group contains the currently installed menu instances. The $objs group for a menu class or menu instance contains a list of menu lines for the menu. A menu can have a $construct() method which is executed when the menu is installed or opened.

Toolbar Classes

A *toolbar class* is a new type of class that defines the contents of a toolbar. A toolbar class contains the buttons and controls for the toolbar and the methods for each control. When you install or open a toolbar using the *Install toolbar* command or the $open() method you create an instance of the toolbar class. The $itoolbars group contains the currently installed toolbars.

You can install an instance of a toolbar into any of the four main docking areas in the OMNIS application window, or toolbars can be floating. You can also install toolbars into a docking area contained in a window class.

Code Classes

A *code class* is a new type of class that you can use as a general code repository. Code classes can contain variables and methods only; they do not have a user interface. You can call the methods in a code class from anywhere in your application using the *Do code method* command. In practice you can put a method in a code class and call it from a

menu or a toolbar: this saves you defining the method twice and is easier to maintain. The $codes group contains all the code classes in your library.

You cannot instantiate a code class and hence code classes do not have instance variables. When you call a method in a code class the instance containing the *Do code method* command remains the current instance. Therefore you can use $cinst in the code class method to refer to the calling instance.

Object Classes

An *object class* is a new type of class that lets you create your own structured data objects. Their structure, behavior, and the type of data they can hold is defined in the variables and methods that you add to the object class.

Object classes can contain custom methods, and class and instance variables. You can create an object variable based on an object class that has the data structure set up in the class. For example, you can create a single instance variable in a window class, based on an object class, which uses the variables and methods defined in the object class. Object classes also support inheritance; you can make a subclass from an object class, which inherits the methods and variables from the superclass.

External Components

OMNIS Studio includes a completely new external components API which supports many different types on component, including ActiveXs, DLLs, OLE-aware objects, and MacOS Code Fragments. You can add external components to the Component Store and use them in your applications. You can get these components from many different sources, including the Internet, but OMNIS provides a few to get you started.

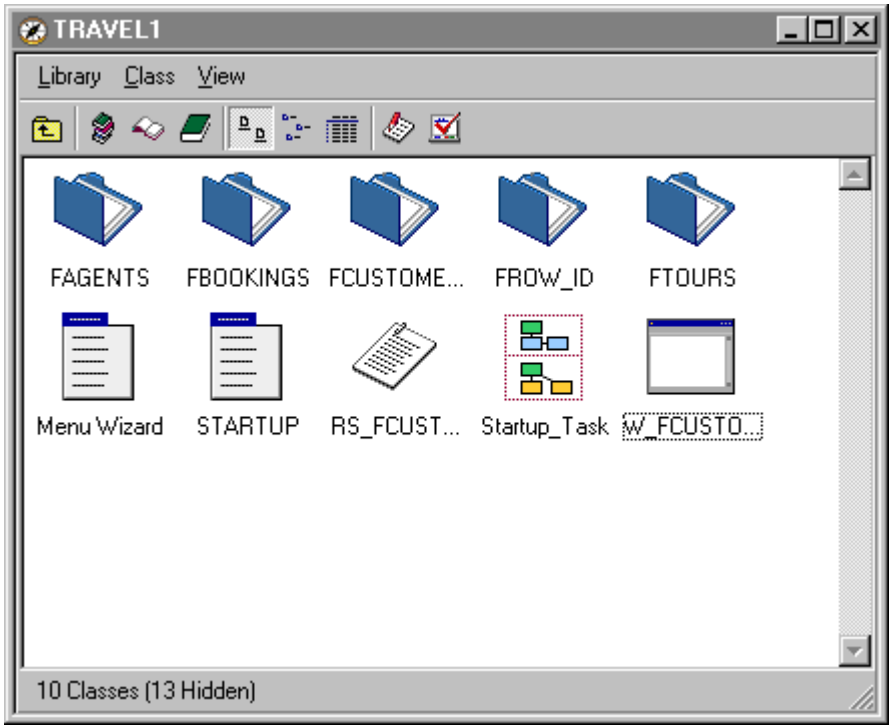
You can also create your own external components; if you use the OMNIS external components API you can use the same code to recompile for the various platforms supported in OMNIS. The OMNIS components API is documented in the *OMNIS External Components* manual, available from the OMNIS website.

New GUI

The OMNIS 7 V3 development environment has been completely redesigned in OMNIS Studio. Some of the tools have been greatly enhanced to take full advantage of the new object technology, while others have been added to make your job of designing an application even quicker and easier. This section introduces some of the new tools and, where possible, compares them with the V3 environment. Note that the development shell available in V3 is removed and replaced by the tools described in this section.

Browser

The V3 Format Browser has been replaced with a Browser that lets you create and examine libraries and classes. Each icon in the Browser represents a single object, either a library or class depending on the current view. You can show multiple instances of the Browser window, which lets you copy classes from one library to another.

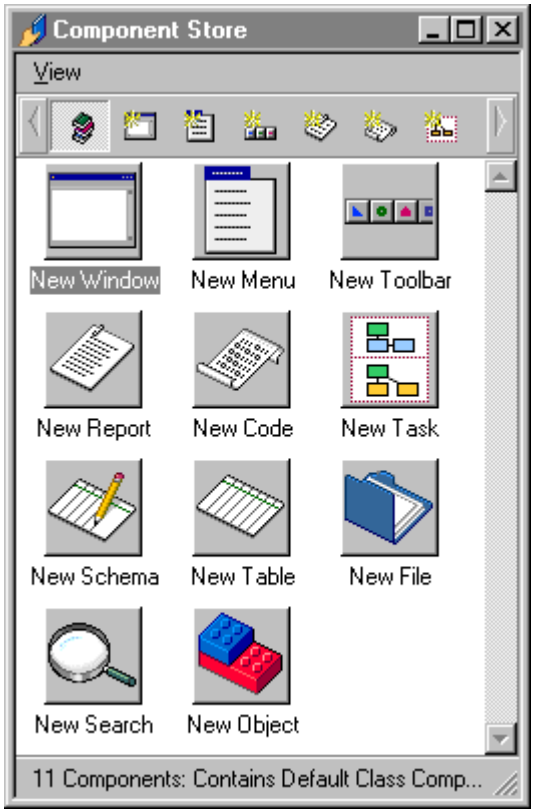


The Browser in large icon mode, showing the classes in the current library

You can show the Browser in large or small icon view, or a detailed view that displays your classes in a table format which you can sort by class, date, class type, and so on.

Component Store

The Component Store is an all-purpose repository for the objects you need to build OMNIS applications. It contains class templates and wizards, field and background objects, and external components. You can create a new object in your library by dragging the object from the Component Store and dropping it onto your library in the Browser. For window classes you can drag fields and other controls from the Component Store onto your window.

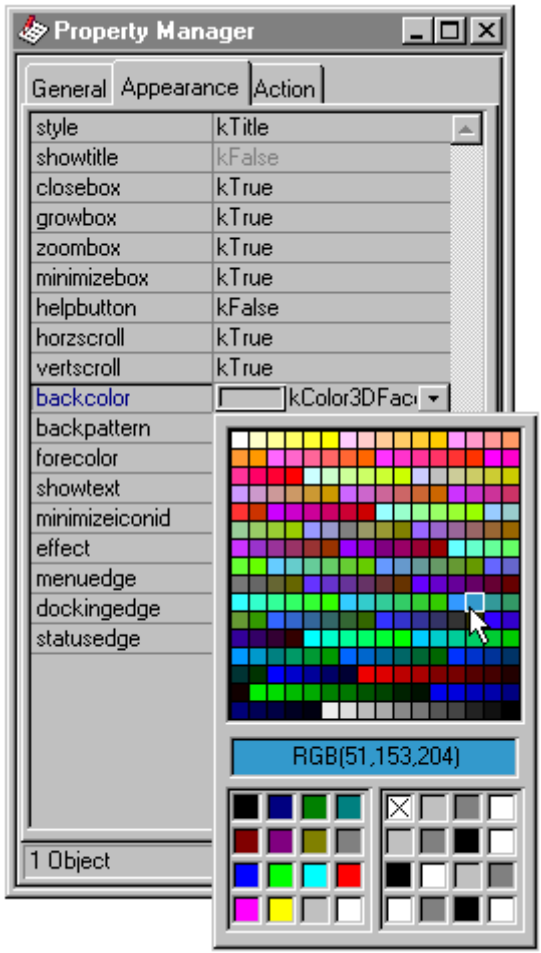


The Component Store showing class templates

The objects in the Component Store are stored in a component library which you can edit to change the base classes in OMNIS. You can also add external components to the Component Store, including ActiveXs, DLLs, and Code Fragments.

Property Manager

The V3 Attribute Inspector has been removed and replaced with the Property Manager and Notation Inspector. You use the Property Manager to display the object properties, including libraries, classes, window fields, and external components. You change the properties of an object by typing a new value into the Property Manager, or by selecting one from a droplist or palette of preset values. You can click on most objects in OMNIS and change their properties in the Property Manager.

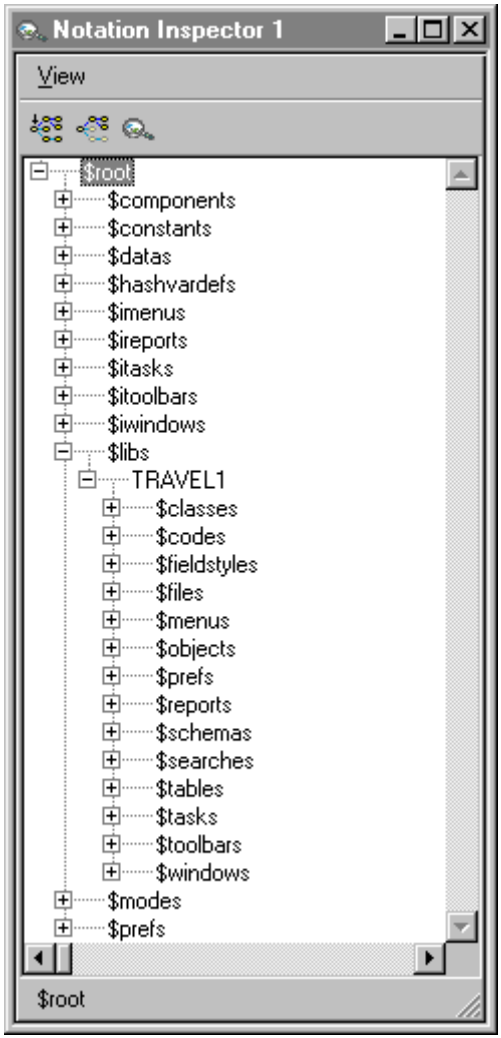


The Property Manager showing a palette of choices for the window background pattern

Notation Inspector

The Notation Inspector lets you view the complete OMNIS object tree from the $root object down. You can expand and collapse each branch of the tree to display the objects in your system or any libraries you have open. The Notation Inspector shows individual objects such as classes and window fields and displays the full notation for these objects. You can click on an object in the Notation Inspector and view its properties in the Property Manager.

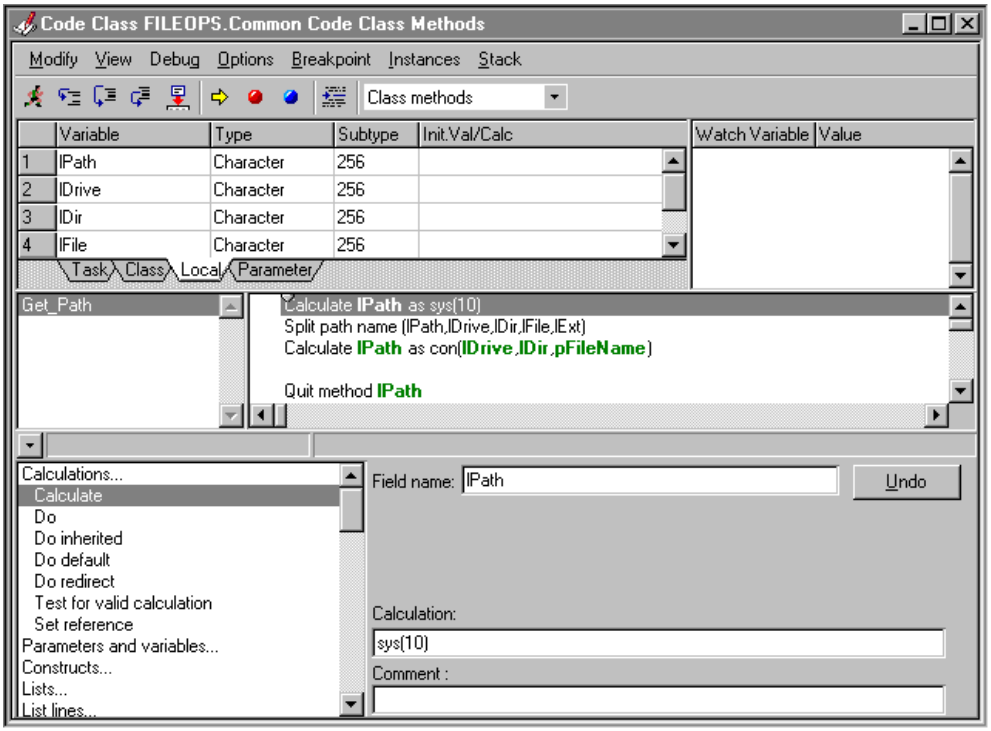
You can drag the notation for an object from the Notation Inspector to a calculation field, and you can copy it to the clipboard. You can also click on an object with the Notation Search tool to view its notation in the Notation Inspector.



The Notation Tree showing the contents of an open library

Method Editor

In OMNIS Studio procedures are renamed *methods*. The procedure editor available in V3 has been greatly enhanced and is renamed the *method editor*. You can add methods to most types of object using the method editor. You also use the method editor to add variables to objects. You no longer declare variables in your code, rather you type their names straight into the variable pane in the method editor. You can inspect the value of any variable or field by Right-clicking on it, and you can monitor values by adding variables to the watch variable pane.

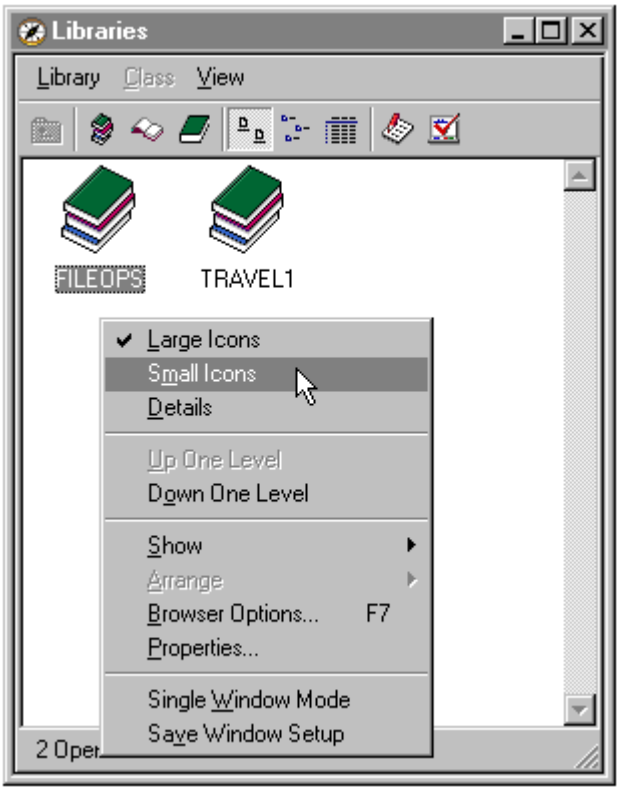


The Method Editor showing a code class method

The method editor has chroma coding, which uses different colors in your code to represent different programming constructs, comments, parameters, and keywords. You use the method editor to create task, code, object, and table classes, and to edit the field and class methods for other types of class including windows and reports.

Context Menus

Context menus are a useful shortcut when you want to modify an object, or change your development environment. You can open context menus from almost anywhere in OMNIS by Right-clicking on an object or window; the term “Right-click” is used throughout the OMNIS manuals and means you click with the right mouse button under Windows, or hold down the Ctrl and click the mouse under MacOS. For example, you can add a method to an object by right-clicking in the method editor, and you can change the view of the Browser using a context menu.

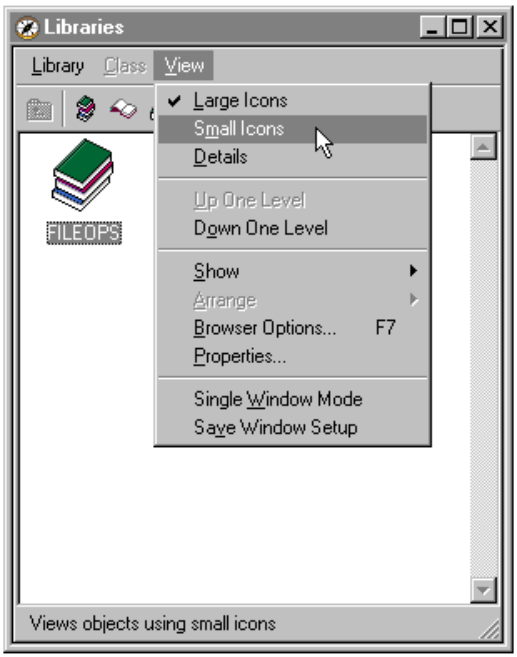


Browser showing its context View menu

The options in a context menu will vary depending on the context of the object or tool you click on, but you will always get the options you need.

View Menus

Most of the design tools in OMNIS Studio have a View menu. View menus are context-sensitive menus that let you change or rearrange the current window. When you change how a tool is displayed you can save its setup using the Save Window Setup option in the View menu for the tool. You can also popup View menus as context menus by Right-clicking on the tool.

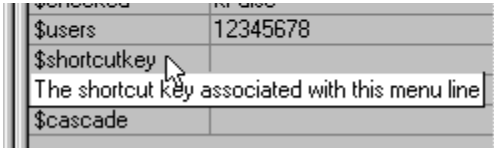


The View menu on the Browser menu bar

Tooltips and Helptips

Many of the design tools in OMNIS Studio have tooltips and helptips. A *tooltip* is a short title or description for a toolbar control, and a *helptip* is a longer description for something like a property in the Property Manager. You can show or hide these tooltips or helptips from the tool's context or View menu. When you move your mouse over a button or object, OMNIS Studio will display the tip.

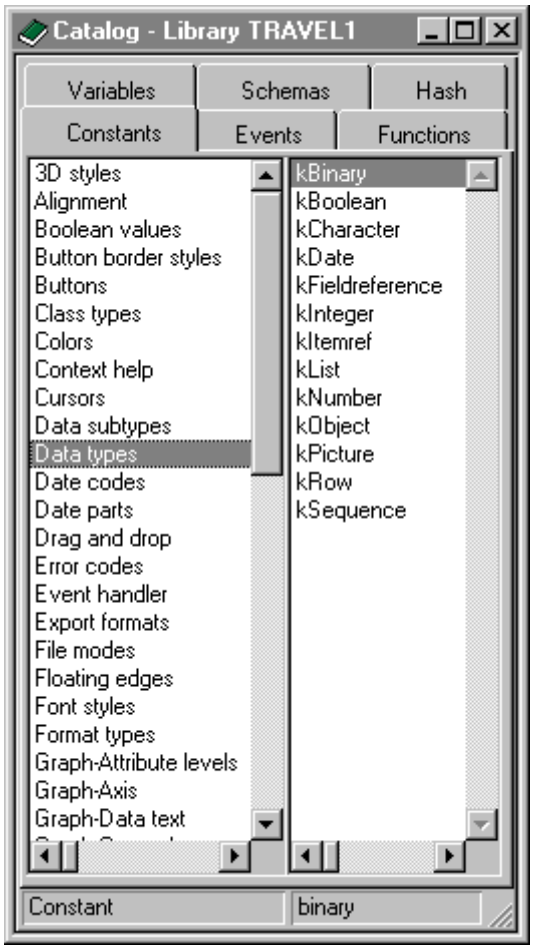
A Tooltip



A Helptip on the Property Manager

Catalog

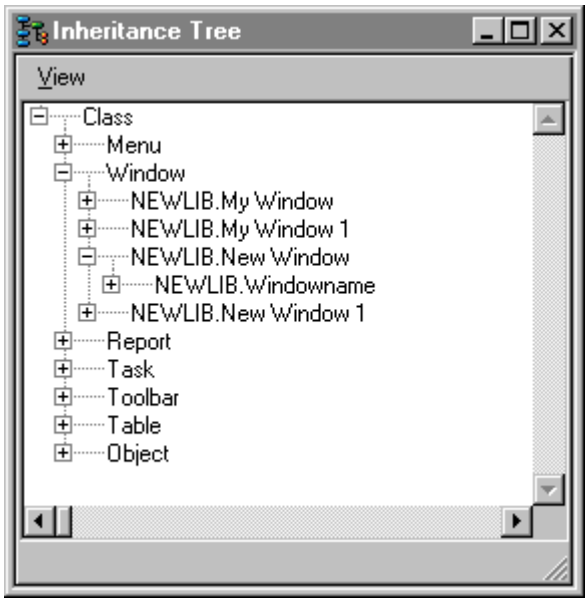
The Field Names window available in V3 is replaced by the Catalog. It lists all the variables and fields in your library, together with the functions, constants, and event messages available in OMNIS Studio. The Catalog also lists the constants and functions for external components. You can open the Catalog by pressing F9/Cmnd-9, as in earlier versions of OMNIS.



The Catalog showing the Data Type constants

Inheritance Tree

OMNIS Studio supports inheritance which lets you create a subclass from another class in your library. The original class is the superclass of the subclass. You can view the superclass/subclass relationships of the classes in your library using the Inheritance Tree.

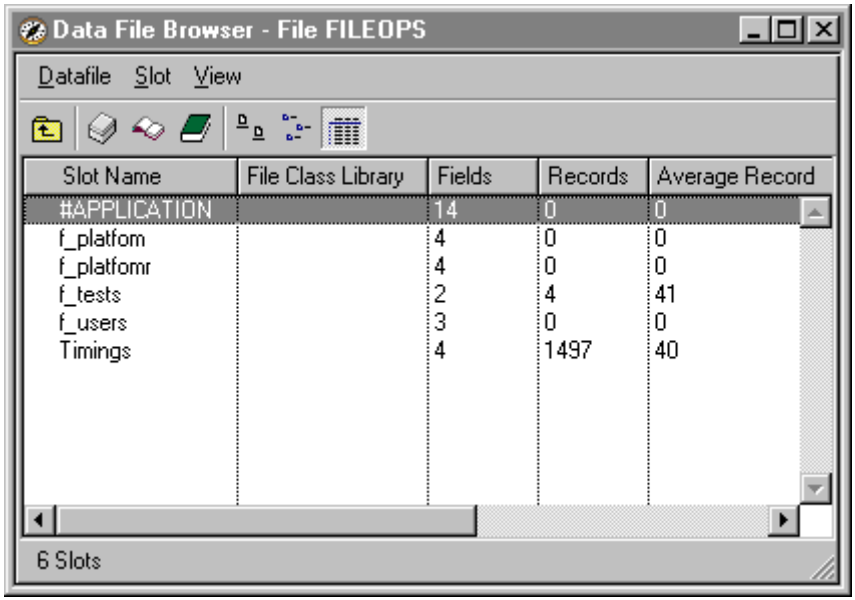


When you select a class in the Browser and open the Inheritance Tree it opens with that class selected, showing its superclasses and subclasses if any are present.

Data File Browser

The Data File Browser replaces various V3 data file tools, including the Data File Tools submenu in the Utilities menu and the old Data File Browser in the development shell. You can view OMNIS data files in the browser with large or small icons or in a detailed view.

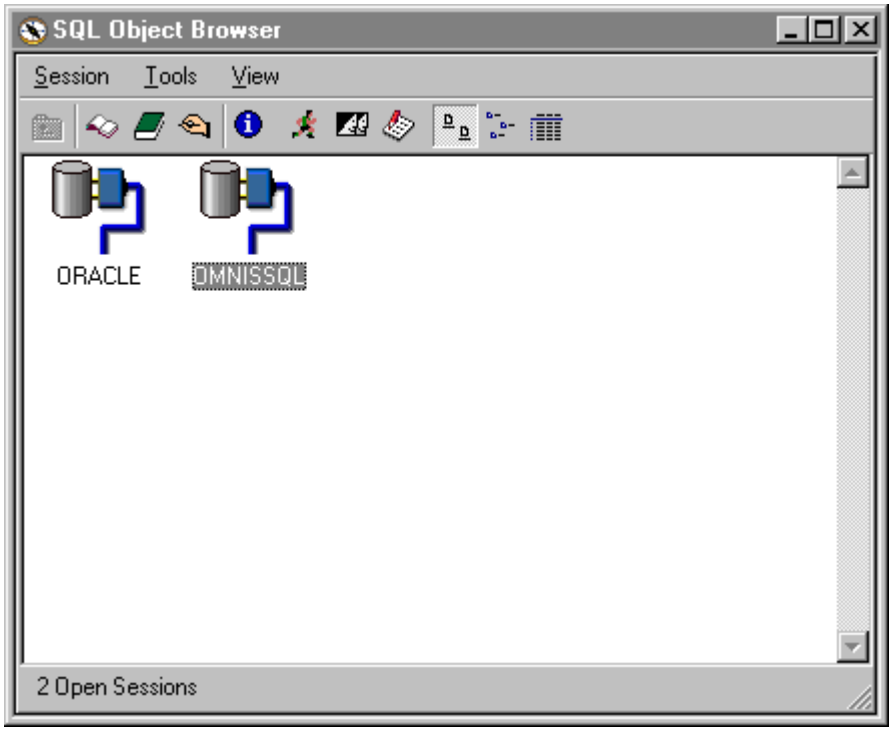
The Data File Browser lets you create new data files, open or close existing data files, and make a particular data file the current data file. It also lets you check and reorganize your OMNIS data, and create, rename, or delete data file slots.



The Data File Browser showing the data slots in an open data file

SQL Object Browser

The SQL Object Browser replaces various SQL tools available in V3 and combines their functionality into one Browser which you can access from the Tools menu. The SQL Object Browser lets you create and open sessions, logon, create new SQL objects such as tables and views, store SQL statements, and perform user administration functions.



The SQL Object Browser showing two open sessions

The SQL Object Browser lets you logon to many different databases including Sybase, Informix, and Oracle. You can also connect to an OMNIS data file using OMNIS SQL. When you are logged on you can view and modify the objects in your database using the SQL Object Browser. See the *Using OMNIS Studio* and *OMNIS Programming* manuals for further details about managing your server or OMNIS database using the SQL Object Browser.

Chapter 2—What's Changed?

This chapter lists the tools, functions, commands, attributes, and other parts of OMNIS that have been renamed, removed, or made obsolete in OMNIS Studio.

What's been renamed?

In OMNIS Studio some existing objects have been renamed while other features have changed to comply with object-oriented terminology or standard industry usage. This section lists all the objects or features that have been renamed or changed.

The major changes are as follows

- formats are renamed *classes*
- procedures are renamed *methods*
- table fields on a window are renamed *complex grids*

These particular changes mean that all objects, attributes, and commands that use these words have also been renamed. For example, $cformat becomes $cclass, the function *fdif()* becomes *cdif()*, $cproc becomes $cmethod, and $tabsection becomes $gridsection.

Other terminology or features have changed, as follows

- under MacOS only, you should Ctrl-click and not Option-click to open variable and context menus
- attributes are separated into *properties* and *methods*, for example, $forecolor is a property, $makelist() is a method; note that methods have parentheses to further distinguish them from properties
- standard attributes, such as $name, are now referred to as *common properties*
- the drag and drop constants kDrop... are renamed *kAccept...*
- list fields you place on windows have been renamed *list box* fields; OMNIS Studio now includes many different list and grid field types
- the parameter variable type Field name has been renamed *Field reference*

- tables on server databases are now referred to as *server tables* to distinguish them from table classes, hence *Describe server table* command
- open windows, installed toolbars and menus, and printed reports are called *instances*, therefore some notation has been renamed or added, for example, $winds becomes $iwindows and contains all the current window instances

For your convenience, the following sections list all the objects that have been renamed. You don't have to do anything within your application to accommodate these changes, OMNIS Studio converts all these objects automatically. However, when you enter literals, such as notation strings, you will have to remember to use the new terminology.

General

Old name	New name (or equivalent)
File format	file class
Window format	window class
Report format	report class
Menu format	menu class
Search format	search class
Format Browser	Browser (displays classes and libraries)
Field Names list	Catalog
Attribute Inspector	Property Manager
procedure editor	method editor
format variables	class variables

Functions

Old name	New name
fdif()	cdif()
fundif()	cundif()

Attributes

Many attributes have been renamed and are now referred to as properties, or in some cases methods. The following table lists the major changes.

Old name	New name	Description
$actnoproc	$actnomethod	property of a pushbutton
$canfocus	$canfocusbuttons	focus for buttons, radio buttons, and check boxes
$cformat	$cclass	the current class, a property of $root
$controlproc	$controlmethod	returns true if the task or class has a $control() method
$controltype	$objtype	the type of toolbar control
$controlwidth	$width	width of the toolbar control
$cproc	$cmethod	the current method
$curlist	$clist	the current list, a property of $root.$modes
$curreport	$creport	the current report, a property of $root.$modes
$cursearch	$csearch	the current search, a property of $root.$modes
$enablemenu	$enablemenuandtoolbars	enables or disables menus and toolbars
$format	$class	class containing the object
$formatdata	$classdata	binary representation of the class
$formats	$classes	object group containing all the classes in the library
$formattype	$classtype	the type of class
$fvardefs	$cvardefs	group of class variable definitions
$fvars	$cvars	group of class variables
$isfloating	$allowdrag	whether or not you can drag a toolbar out of a docking area
$proc	$method	the method containing the object
$procs	$methods	group of methods for the object: classes, fields, and other objects can now have multiple methods
$proctext	$methodtext	all command lines for the method returned as text

Old name	New name	Description
$showselected	$multipleselect	if true shows multilple selected list lines
$table	$grid	grid field containing the object
$tabsection	$gridsection	grid section containing the object
$winds	$iwindows	group of window instances

Constants

The following constants have been renamed.

Old name	New name
kcursDrag	kcursDragObject
kDropAll	kAcceptAll
kDropButton	kAcceptButton
kDropComboBox	kAcceptComboBox
kDropDroplists	kAcceptDropList
kDropEdit	kAcceptEdit
kDropPicture	kAcceptPicture
kDropPopMenu	kAcceptPopMenu
kDropStdList	kAcceptList
kDropTable	kAcceptGrid
kerrBadproctext	kerrBadmethodtext
kFieldname	kFieldreference
kListobj	kListBox
kNoDrop	kAcceptNone
kTabcolheader	kGridcolheader
kTabheader	kGridheader
kTable	kSystemtable
kTableobj	kComplexgrid
kTabnone	kGridnone
kTabrest	kGridrest
kTabrow	kGridrow

Note that the constant kTable which represents an OMNIS system table has been renamed kSystemtable. There is a new constant called kTable which represents the new table class. Also kTableobj which represents a window field type has been renamed kComplexgrid.

Commands

The following commands have been renamed.

Old name	New name
Call procedure	Call method
Clear format variables	Clear class variables
Clear procedure stack	Clear method stack
Clear search format	Clear search class
Clear timer procedure	Clear timer method
Close print file	Close print and export file
Delete format	Delete class
Duplicate format	Duplicate class
Make file format from table	Make file class from server table
Modify format	Modify class
Modify procedures	Modify methods
New format	New class
Open window	Open window instance
Optimize procedure	Optimize method
Print format	Print class
Prompt for print file	Prompt for print and export file
Prompt for search format	Prompt for search class
Quit all procedures	Quit all methods
Quit procedure	Quit method
Reinitialize search format	Reinitialize search class
Rename format	Rename class
Revert format	Revert class
Save format	Save class
Set ‘About...’ procedure	Set ‘About...’ method
Set format description	Set class description
Set print file	Set print and export file
Set timer procedure	Set timer method
Test data with search format	Test data with search class

External Commands

Generally, the names of external commands have been split into separate words to make them more readable.

Old name	New name
CallDLL	Call DLL
CallXCMD	Call XCMD
ChangeWorkDir	Change working directory
CloseFile	Close file
CopyFile	Copy file
CreateDirectory	Create directory
CreateFile	Create file
DeleteFile	Delete file
DIROPS Get Files	Get files
DIROPS Get Folders	Get folders
DoesFileExist	Does file exist
GetFile	Get filename
GetFileInfo	Get file info
MoveFile	Move file
OpenFile	Open file
PutFile	Put filename
ReadFile	Read file as character
RegisterDLL	Register DLL
SetCreatorType	Set creator type
SplitPathname	Split pathname
WriteFile	Write file as character

Note ReadFile and WriteFile have been renamed to *Read file as character* and *Write file as character*, and *Read file as binary* and *Write file as binary* have been added to supplement these commands.

What’s been removed?

This section describes the objects and features available in OMNIS 7 V3 that have been removed or are no longer supported in OMNIS Studio. Some items have been removed without replacement, while others have been superseded by more streamlined functionality.

General Features

External Areas and Graphs

The *external area* field type has been removed in OMNIS Studio. However, you can add external objects to your environment using the new external components interface. For example, in V3 you could place a graph on your window using an external area, but in OMNIS Studio graphs are implemented as external components, available in the Component Store.

Ad hoc Reports

Ad hoc reports have been removed from the core product and replaced by a separate OMNIS library which you can load and use like any other library. All commands and notation associated with the old ad hoc report engine have also been removed. The new ad hoc report library will convert your old ad hoc report files, although some of the text formatting will be lost.

Publish and Subscribe

Support for Apple’s *Publish and Subscribe* has been removed in OMNIS Studio. All commands and notation associated with publish and subscribe have also been removed.

Color System Table

The #COLORS system table and all corresponding notation have been removed. In OMNIS Studio an object stores its color as an RGB value. The color constants have specific RGB values. Generally you should use the color constants to set colors, but you can use the *rgb()* function to set the color of an object.

Fixed Column Lists

The *Store Long Data* option has been removed from the *Define list* command. Therefore all lists store unlimited lengths of data by default, and each column is only restricted by its data type and length defined in the variable or field representing the list column. You can still restrict the length of an individual column using Variablename/Length in the *Define list* command, but this notation does not work for the new $define() or $definefromtable() list methods.

Design Menu

The *Design* menu has been removed and replaced by the Tools and View menus. Also context menus replace many of the Design menu features. To popup a context menu, right-click on the area you're working in.

Design and SQL Shell

The *Design Shell* has been removed, but some parts of the shell, such as the VCS, are available in OMNIS Studio as separate products.

The *SQL Shell* has been removed in its OMNIS 7 V3 form and is replaced by a new library that includes the SQL Object Browser and Logon Manager.

Tool Palette and Field Attributes Tool Box

In V3 when you created a field or double-clicked on a field the tools palette and field attributes tool box opened. These tools no longer exist and have been replaced by the Component Store and Property Manager. For example, you used to change the color of an object in the tools palette which you now do via the Property Manager. You create different types of field from the Component Store and you change the properties of an object in the Property Manager. If you double-click on a field in OMNIS Studio the method editor opens which lets you add methods to the object.

Mixed Case Field Names

The Mixed case field names library preference $mixedfieldnames has been removed. All file class field names can be mixed case by default.

Modify Menus

The *Modify* menus for classes (formats) have been removed from the main menu bar. In many cases these menus now appear on the appropriate class editor window. For example, the Modify method menu is now available in the method editor menu bar. Context menus also replace menu items in the old Modify menus.

Help Data File

The *Help data file*, which supplied help for menu lines and other interface elements, has been removed. Also the *Help message* command has been removed. However, most visible objects such as window fields and menu lines contain the helptext property in which you can enter help messages. When you convert your application help messages are transferred to the helptext property in the converted field.

Toolgroups

Toolgroups have been removed in OMNIS Studio. Groups of toolbar controls are no longer stored with a window or menu, rather they are stored in separate toolbar classes. When you convert your application toolgroups are removed from their old window or

menu format and converted to toolbar classes. Much of the old toolgroup notation has been removed and replaced by more generic class notation.

Table Fields

Table fields have been renamed complex grids, and much of the old table field notation has either been renamed or removed. In most cases grid field notation has been replaced by more generic object notation. For example, $tabrobjs the group of table row objects has been removed and replaced by $objs containing all the objects in the grid field. The $rowsection property and #INSIDE have also been removed from grid fields.

Split Bars

Split bars have been removed from window classes, including the properties $hsplitbar and $vsplitbar.

Format Header Object

The $head object has been removed from all classes (formats), including window and report classes. All objects and notation that appeared under the header object in V3 now appears directly under the class itself. Therefore when you reference window and report objects using the notation you no longer need to include $head. The header object for file classes is also removed, but $datahead is retained for file classes.

MSMail

Support for MSMail has been removed, including all MSM external commands.

Hash Variables

The following hash variables are no longer supported in OMNIS Studio and do not appear in the Catalog. For converted libraries only, these hash variables and the V3 event mechanism continue to work provided the library preference $v3events is set to kTrue. For new OMNIS Studio libraries you should write event handling methods using the new event codes. See the Catalog or *OMNIS Help* for a complete list of event codes.

Hash variable	Equivalent event code
#AFTER	evAfter
#BEFORE	evBefore
#BEFORE1	None
#BEFORE2	None
#CANCEL	evCancel
#CLICK	evClick
#CLOSE	evClose
#DCLICK	evDoubleClick

Hash variable	Equivalent event code
#DELETE	evStandardMenu
#DISABLED	evDisabled
#DRAGREF	event parameter
#DROP	evDrop
#EDATA	None
#EDIT	evStandardMenu
#EF	None
#EFLD	None
#EM	None
#EN	None
#ENABLED	evEnabled
#ER	None
#FIND	evStandardMenu
#HIDDEN	evHidden
#HOLD	evMouseHold
#HSCROLLED	evHScrolled
#INSERT	evStandardMenu
#INSERTCV	evStandardMenu
#INSIDE	None
#KEY	evKey
#KEYPRESS	evKeyPress
#LCHANGED	evRowChanged
#MAXIMIZED	evMaximized
#MFIELD	None
#MINIMIZED	evMinimized
#MODIFIED	None
#MOUSEDOWN	evMouseDown
#MOUSEENTER	evMouseEnter
#MOUSELEAVE	evMouseLeave
#MOUSEUP	evMouseUp
#MOVE	evMoved
#NEXT	evStandardMenu

Hash variable	Equivalent event code
#OK	evOk
#PREVIOUS	evStandardMenu
#RESIZE	evResized
#RESTORED	evRestored
#RHOLD	evRMouseHold
#RMOUSEDOWN	evRMouseDown
#RMOUSEUP	evRMouseUp
#SENT	evSent
#SHOWN	evShown
#SKEY	evKey
#STAB	evStab
#VSCROLLED	evVScrolled
#WCLICK	evWindowClick

Attributes

Many attributes have been removed, some without replacement. The following table lists the important ones.

Attribute	Alternative, if available...
$3dInterface	None; all windows and buttons have a 3d interface
$allowdrop	None
$ansiforascandchr	None
$autonameswindow	the Field Names window is replaced by the Catalog which does not pop up when required as before, you press F9/Cmnd 9 to open the Catalog
$autopubs and $autosubs	Publish and Subscribe and all associated notation has been removed
$autoscroll	None
$blockcarets	None; caret in picture fields is replaced with dotted line
$checkboxsize	None
$colheader	$showheader, $showhorzheader, $showvertheader
$colorlookup	None
$colormap	None

Attribute	Alternative, if available...
$colors	None
$controlcallproc	$methods; toolbar controls can now contain multiple methods
$controlinitchoice	None
$controlinitdisable	None
$controllist	$dataname of the list
$controllistcalc	$calculation for toolbar control
$controlpos	objects in toolbar are listed in $objs for the class
$controls	$objs contains all the objects in toolbar class
$datahead	the data header object has been removed from all classes except file classes; its contents are now contained in the class itself
$dateformats	edit the #DFORMS table directly
$decimalpoint	None
$defaultsizes	None
$defaultwindowcolor	None; all windows and buttons have a 3d interface
$dlib	$designtaskname sets the design task name
$fieldname	$name is the object name of the field, $dataname is the name of the field or variable for the field
$filedotfieldseparator	None
$floating	floating report fields have been removed
$functionseparator	None
$groupname and $groups	$name for a toolbar class, $itoolbars lists the installed toolbars in the main docking areas
$hasfocus	$cfield is the current field
$head	the header object has been removed from all classes; its contents are now contained in the class itself; $datahead for a file class remains
$hsplitbar	window split bars have been removed
$isexport	$exportformat sets the data format; default is kEXnone which means the report does not export its data
$libvars and $libvardefs	library variables have been removed and replaced by task variables contained in $tvars (task variables)
$mixedfieldnames	all fields now have mixed case names by default

Attribute	Alternative, if available...
$noadhocs	$loadadhocs OMNIS preference that returns whether or not your serial number allows access to ad hoc reports
$newcontrolbehavior	None
$pixheight	use $height of the object
$protected	None; you can no longer protect a class
$radiobuttonsize	None
$reportfonts	use field styles to implement cross-platform fonts
$rowsection	use a group field or scroll box to implement scrollable field
$thousands	None
$showhelpcodes	the help data file has been removed
$state	$checked sets the checked status of a toolbar control
$tab..., $tabc..., $tabh..., and $tabr... (table field notation)	notation for grid field, including $objs and $bobjs containing the fields and background objects in the grid field
$toolbar	the group of OMNIS toolbars has been removed from the $root preferences
$toolgroup	toolbars are separate classes
$toolowner	None
$v2combos	None
$visondonotgray	None
$vsplitbar	window split bars have been removed
$windowfonts	use field styles to implement cross-platform fonts

Constants

The following constants have been removed.

Old constant	Alternative, if available...
kExternalarea	external areas have been removed; use external components

Commands

Several commands have been removed in OMNIS Studio. Some commands are commented out with the message COMMAND REMOVED BY CONVERTER while others are removed altogether. The following commands have been removed.

Old command	**Alternative usage, if available...**
Call external with return value	*Call external Returns* command
Call procedure with return value	Converted to *Call method Returns* command although you should use *Do method*
Clear format variables when closed	*Clear class variables* in $close() method
Do not clear format vars when closed	None
Do not pass event	*Quit event handler Discard event* command
Format variable	Declare class variable in method editor
Help message	$helptext property for menus and menu lines or $tooltip for toolbar and window objects
Install hierarchical menu	$cascade property for a menu line
Library variable	Declare task variable in method editor
Local variable	Declare local variable in method editor
New ad hoc report	Ad hoc reports are now an OMNIS library
Open window maximized	*Open window instance /MAX* command
Open window minimized	*Open window instance /MIN* command
Other parameters are optional	None
Parameter	Declare parameter variable in method editor
Print ad hoc report	Ad hoc reports are now an OMNIS library
Quit procedure (flag clear)	*Quit method returns kFalse* command
Quit procedure (flag set)	*Quit method returns kTrue* command
Quit to enter data	*Quit event handler* command
Replace standard Design menu	None (the Design menu has been removed)
Set design library	$designtaskname for the class

What is Obsolete?

Objects or commands that are marked OBSOLETE in OMNIS Studio are objects that are no longer supported in this version and are included for converted libraries only; they will be removed in the next major release of the product. When creating new libraries, you should not use anything marked OBSOLETE. You should use an alternative command or function, or if posssible use the equivalent notation.

Constants

The following data export formats have been removed without replacement. Use one of the remaining export formats, such as tab-delimited.

Constant
kEXdBase_OBSOLETE
kEXdif_OBSOLETE
kEXlotus_OBSOLETE
kEXsylk_OBSOLETE

Attributes

A few properties (attributes) have been made obsolete. Obsolete properties are marked “$was_oldname” where oldname is the old name of the attribute. For example, you may see the property $was_visondonotgray in converted libraries.

Commands

In V3, some commands, such as *Enable fields* and *Queue click*, used to take a *field number* as a parameter. These commands have been made obsolete and replaced with equivalent commands that take a *field name* or list of field names as a parameter. Where possible you should exchange the obsolete commands for the new ones.

The following table lists the commands that have been made obsolete. They appear in the command list in the method editor under the Obsolete commands group.

OBSOLETE command	**Alternative usage, if available...**
Call method (was Call procedure)	*Do method* or *Do code method* command
Clear task control method (was Clear library control procedure)	rename/remove task $control method
Clear window control method (was Clear window control procedure)	rename/remove window $control method
Disable fields *field-number*	Disable fields *field-name(s)*
Enable field *field-number*	Enable field *field-name(s)*
Hide design & commands menus	Design menu has been removed
Hide fields *field-number*	Hide fields *field-name(s)*
Load page setup	None
Make file class from server table (was Make file format from table)	None
Map fields to host	*Fetch next row var1,var2,...* command
Queue click *field-number*	Queue click *field-name*
Queue double-click *field-number*	Queue double-click *field-name*
Queue scroll *field-number*	Queue scroll *field-name*
Queue set current field *field-number*	Queue set current field *field-name*
Redraw named fields	*Redraw* command or $redraw method
Redraw numbered fields	*Redraw* command or $redraw method
Redraw windows	*Redraw* command or $redraw method
Send to a window field *field-number*	Send to a window field *field-name*
Set return value	Quit method Returns *return-value*
Set task control method (was Set library control procedure)	enter $control method in task class
Set window control method (was Set window control procedure)	enter $control method in window class

OBSOLETE command	**Alternative usage, if available...**
Show fields *field-number*	Show fields *field-name(s)*
SNA do not perform default action	Quit event handler *Discard event*
SNA perform a Cancel	Quit event handler, Queue cancel
SNA perform a shift-tab	Quit event handler, Queue tab (Shift)
SNA perform a tab	Quit event handler, Queue tab
SNA perform an OK	Quit event handler, Queue OK
SNA perform command	Quit event handler *Discard event*, Call method
SNA perform default action	Quit event handler *Pass to next handler*
SNA remain on current field	Quit event handler *Discard event*
SNA set current field	Quit event handler *Pass to next handler*, Do $ctarget.$assign(NewTargetField)
Store window	None
Test if command available	None

Chapter 3—Conversion and Compatibility

This chapter describes how you convert your application created with OMNIS 7 Version 3 or an earlier version and discusses the compatibility of converted libraries in OMNIS Studio.

The vast majority of OMNIS applications will convert to OMNIS Studio without error. However, some complex applications will require attention following conversion. The trace log reports any errors during conversion and you can use the information in this chapter to fix your libraries.

Converting your old OMNIS Applications

OMNIS 7 Version 2 or earlier

If you want to convert an application created with OMNIS 7 Version 2 or earlier you must convert it to OMNIS 7 Version 3 first and then open it in OMNIS Studio. There is a demonstration copy of OMNIS 7 Version 3 supplied with OMNIS Studio that you can use to convert your old applications to V3. You can convert OMNIS 5, OMNIS 7 Version 1, and OMNIS 7 Version 2 libraries to V3. Having converted your library to V3, you can convert your library to OMNIS Studio as described in this chapter.

OMNIS 7 Version 3

If you want to convert an application created with OMNIS 7 Version 3 you simply open it in OMNIS Studio. OMNIS 7 Version 3 libraries are converted to OMNIS Studio automatically.

WARNING When you open a V3 library in OMNIS Studio, all library files and data files are converted to OMNIS Studio, and all V3 formats are converted to OMNIS classes. After conversion *you cannot use these converted libraries and data files with earlier versions of OMNIS*. Therefore, make sure you have a secure backup of your application before you convert it to OMNIS Studio.

Converting Libraries and Data Files

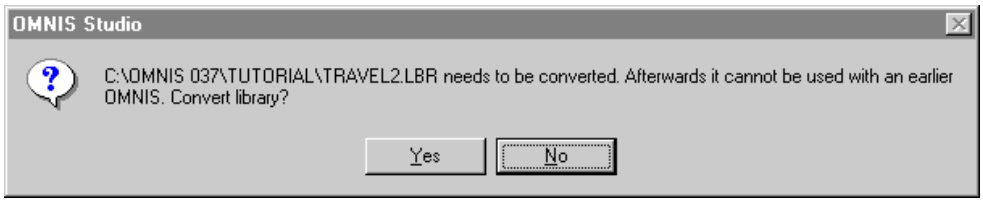
To convert a library created with OMNIS 7 Version 3 you simply open it in OMNIS Studio.

To convert an OMNIS 7 V3 library

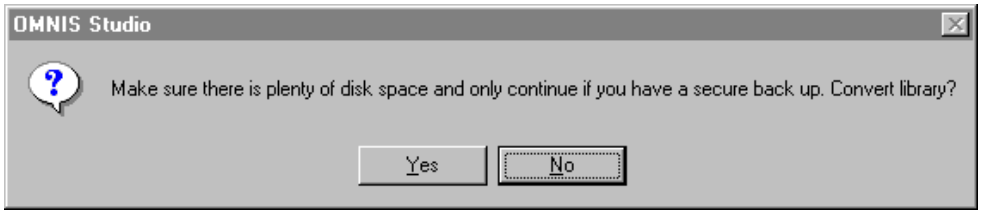
- Make a backup copy of your V3 library
- Start OMNIS Studio and open the Browser
- Select the Library>>Open option in the Browser menu bar and open your library

You cannot drag a V3 library icon onto the OMNIS Studio program icon to open and convert the library. You must convert a library by opening it in OMNIS Studio using the Library>>Open option in the Browser. Following conversion you can drag a library icon onto the program icon to open the library.

When you attempt to open a V3 library in OMNIS Studio you will get the following message



If you answer Yes to this message and continue the conversion process you will get another message



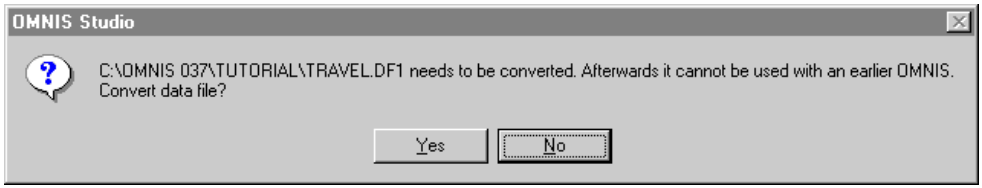
The conversion process is complex and it is essential that you have a secure backup copy of your library before you convert it. Furthermore, after conversion your library file may be up to 60% larger than before, therefore you need sufficient disk space to accommodate your converted library. Note that small V3 libraries may not grow at all.

If you answer Yes to the second message and proceed with conversion, OMNIS Studio starts to convert your whole library. While your library is being converted OMNIS Studio shows you its progress, and displays the Trace Log recording any conversion errors (see below). For small libraries, containing only a few formats, conversion will not take long. However for large complex libraries OMNIS Studio may take several minutes to convert your library.

Note that OMNIS Studio will convert a V3 library that is locked, without warning. However it will not alert you that the library is locked until it has finished converting the library.

Converting Data Files

OMNIS Studio will convert data files created with OMNIS 3 Version 3 automatically. When you open a V3 data file, or your converted library attempts to open an old data file, you will get the following message



It is essential that you have a secure backup copy of your data file before you convert it. A very large data file may take several minutes to convert and may require reorganizing.

Old Data Formats

If you have an old data file created with OMNIS 3+, OMNIS Quartz, or OMNIS 5 you *must* convert it to OMNIS 7 Version 3 data format before loading it in OMNIS Studio. To do this, load your old data file in the demonstration copy of OMNIS 7 Version 3 supplied with OMNIS Studio.

Trace Log

When you convert a library, OMNIS Studio will open the trace log to record any errors during conversion. Where an error or problem occurs, the converter will insert a comment “CONVERTER ERROR: Could not convert...” and the trace log will tell you to “Search class for CONVERTER ERROR to find problem(s)”. In this case, you can use Find and Replace under the Edit menu to search the specified class for “CONVERTER ERROR”. Note that you can double-click on a line in the Trace Log to open the class containing the error, and similarly, you can double-click on a line in the Find and Replace Log to go to the specified item, including a method line containing the error.

Viewing the Contents of your Converted Library

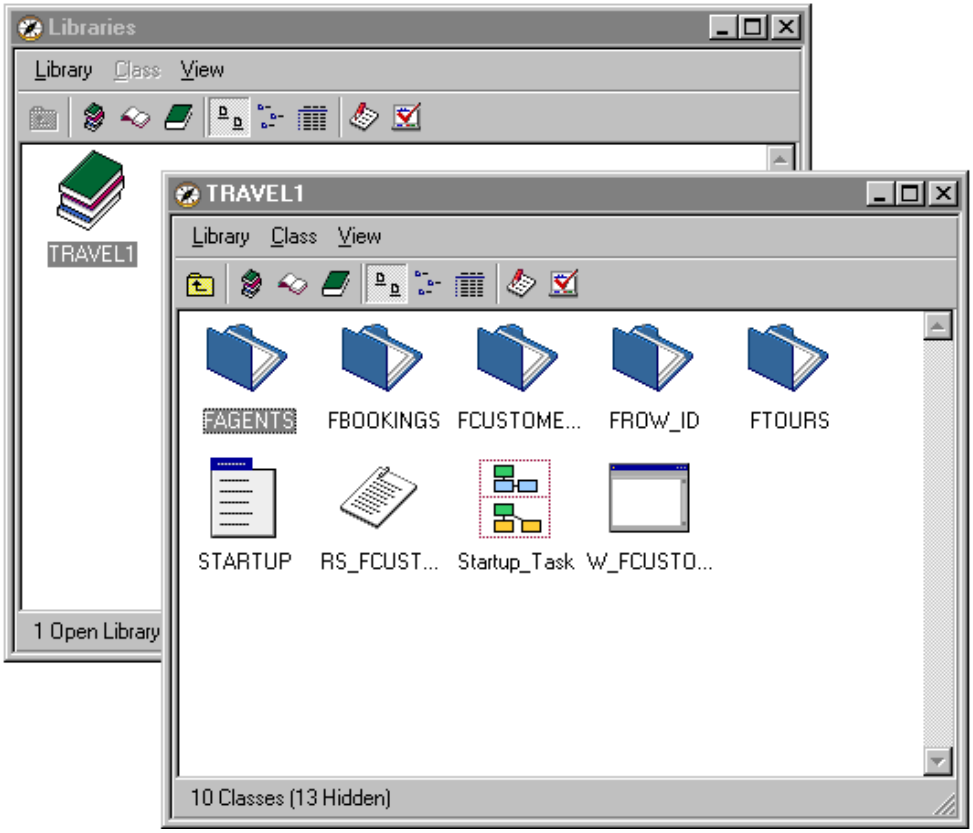
When OMNIS Studio has finished converting your library it will appear in the Browser as a single icon.

To view the classes in your library

- Double-click on the library icon

or

- In the Browser, select your library and choose the View>>Down One Level menu item from the Browser menu bar, or click on the Down One Level button



Another Browser window opens showing the contents, the classes, in your converted library. You can hide and show different types of class using the Browser Options available under the View menu.

To change the Browser options

- Select the View>>Browser Options menu item on the Browser menu bar, or press F7/Cmnd-7

You can select Include All types of class in the Browser, or click the Exclude All and check only those classes you want to display. Note that the default is to display all the available classes, except the system tables.

Default Internal Library Names

When you open your library in OMNIS Studio the disk file name of the library becomes the *default internal library name.* When you convert a library the disk file name becomes the internal library name of the converted library.

You use the default internal library name to reference classes outside the current library using the notation LIBNAME.CLASSNAME. Even if you rename your library file on disk the default internal name is unchanged. However, you can assign a new name to the $defaultname library preference.

Format Conversion

When you convert a library to OMNIS Studio all V3 formats are converted to the new OMNIS class types automatically. For example, a V3 file format will convert to a new file class, and a report format will convert to a new report class.

If your old library contains a STARTUP menu, OMNIS Studio converts it to a new menu class called STARTUP and adds a task class called Startup_Task to your library. The startup task is run automatically when you open your library and installs the STARTUP menu.

If your old library contains toolgroups, these are removed by the converter and replaced with new toolbar classes with the name T_OLDWINDOWNAME or T_OLDMENUNAME depending on whether they were associated with a window or menu format.

Please refer to the remainder of this chapter for details about specific types of format and library objects and how they convert in OMNIS Studio.

Format Data

You must convert V3 libraries as a whole: you cannot convert individual V3 formats to the new class types by copying a V3 format to an OMNIS Studio library. The notation $classdata.$assign(), and functions cdif() and cundif() will generate an error if the source of the data is a V3 format.

Format and Class Names

OMNIS Studio does not allow trailing spaces in class names, and notation containing such names will fail. Therefore you should strip out the trailing spaces. Generally you should avoid putting spaces and non-alphanumeric characters in all class names.

Menu Formats

When you convert a V3 library to OMNIS Studio each menu format is converted to a new menu class of the same name. The new menu class contains two sets of methods: Line methods and Class methods. The procedures in the old menu format become Class methods in the new menu class. Each Line method in the new menu class calls the appropriate class method.

The Procedure line 0 in the old menu becomes the $construct() method in the new menu class, which is executed when the menu is installed. The new menu class has the same number of menu lines as the original menu format and the text and other properties, such as initially checked, and initially enabled, also remain the same. The format variables in the old menu format become class variables in the menu class.

Calls by name to Procedure 0 of an old menu now fail because the method is now called $construct().

The *Disabled if no current record* and *Quit all when selected* menu line options are removed without replacement.

With previous versions there were times when a menu was installed and Procedure 0 was *not* called, for example, if it was installed as a hierarchical menu. This is no longer the case, the $construct() method is always executed whenever a menu is installed, that is, a menu instance is constructed. For example, the $construct() method of hierarchical and context menus is called when the instance is created.

Call procedure command

In V3 you may have used menu procedures as general-purpose procedures that you called from elsewhere in your library. In OMNIS Studio the *Call procedure* command has been renamed *Call method* and is marked as OBSOLETE, but continues to work in converted libraries. The command now uses a method name and not a procedure number as before. All *Call procedure* commands which used a procedure number are changed wherever possible to *Call method MethodName,* although some failures may occur as follows: where a method in a different library is called, when indirection is used, where ambiguous names are involved, where an old window procedure has no name and was called by number alone, and where a numeric name or a name containing special characters is used. In these special cases the original *Call procedure* command will now fail.

It is also possible that your converted library will contain two or more methods with the same name, although in V3 they may have had different procedure numbers.
OMNIS Studio converts all procedure numbers to method names so you will have resolve duplicate names manually.

Procedure Numbers

To help you debug a converted library you can show the V3 procedure numbers using the View>>Show V3 Method Numbers menu option in the method editor menu bar. This option places a number in parentheses next to each method name. This facility is for converted libraries only; you should not use method numbers to call methods, use method names.

Menu Commands

In OMNIS Studio you must specify a menu instance name and not the class name for most menu commands. When you use the *Install menu* command you specify the name of the menu class to be installed and you can assign an instance name. If you omit the instance name it defaults to the class name.

The *Replace standard menu, Remove menu, Test for menu installed, Enable menu, Disable menu, Enable menu line, Disable menu line, Check menu line,* and *Uncheck menu line* commands all take an instance name and not a class name.

References to all standard menus beginning with an asterisk, such as *File, are no longer supported. Therefore all menu commands that refer to standard menus will be commented out. For example *Install menu *Utilities* will be commented out with the line COMMAND REMOVED BY CONVERTER.

The *Install hierarchical menu* command has been removed and the converter will comment out any occurrences. It has been replaced by the $cascade property which is a property of a menu line. The converter will set the $cascade property to the menu class name for the appropriate menu line. To view or set the $cascade property, open your converted menu class, click on the menu line that contained the hierarchical menu, open the Property Manager. The **cascade** property appears under the General tab for the menu line.

The *Build installed menus list* command now builds a list of current menu instances installed on the main menu bar.

The $construct() method of any menu, including the startup menu, is always called when the menu is instantiated. Therefore the “Do not call startup procedure” option has been removed from the *Open library* command.

Menu Notation

LIBRARY.$menus contains all the menu classes in your library, and $root.$imenus contains all the current menu instances, listed in the order they appear on the main menu bar. Hierarchical, popup, window menus, and context menus are not included in $imenus.

A menu class and a menu instance have the $objs group containing the menu lines in the menu class.

STARTUP Menu and Startup Task

When you convert a V3 library containing a STARTUP menu, the menu format is converted to a new STARTUP menu class, and a new task class called Startup_Task is added. These new classes combine to give the same functionality as the old STARTUP menu format.

The procedures in the old STARTUP menu format are converted into methods and placed in the new STARTUP menu class. The procedures in the old menu format become Class methods in the new menu class. Each Line method in the new menu class simply calls the appropriate class method.

The Startup_Task is a special type of class. A startup task is opened automatically and the $construct() method is executed when you open a library. In converted libraries the $construct() method of the Startup_Task contains an *Install menu STARTUP* command that installs the new STARTUP menu class. You can put any code you want to run when your library is opened in the $construct() method of the startup task.

Library Control Procedures and Tasks

Library control procedures are no longer supported and are replaced by task $control() methods. The *Set* and *Clear library control procedure* commands are renamed to *Set* and *Clear task control method* and are marked as OBSOLETE. This causes few compatibility problems if you create a single startup task for your library and open all instances in this task. However the **All libraries** and **No windows open** options have been removed which may cause compatibility failures when you convert your library.

Library and Format Variables

When you convert a V3 library to OMNIS Studio, all library variables are converted to *task* variables, and all format variables are converted to *class* variables. Local and Parameter variables are unchanged and are inserted in the appropriate method.

The *Library variable*, *Format variable*, *Local variable* and *Parameter* commands have been removed without replacement. You no longer declare variables in your code using a

command, rather you add a variable name and definition to the variable pane in the method editor. Variables defined in procedures in your old library are removed and listed in the variable pane for the appropriate class or method.

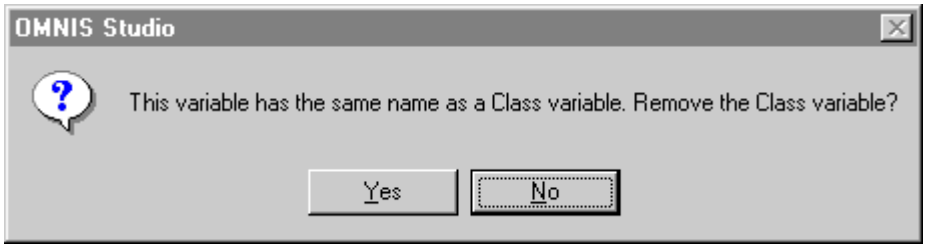
The *Clear format variables when closed* and *Do not clear format vars when closed* commands have been removed without replacement. When you convert your library they are commented out by the converter. However you can use a *Clear class variables* command in the $close() method for a window to clear class variable values.

Variables Notation

The notation groups $libvars and $libvardefs have been removed. The notation groups $fvars and $fvardefs are renamed $cvars and $cvardefs. Conversion only happens for tokenized code and not for literals. For example, if $fvars occurs in a literal within square brackets it will not be changed by the converter.

Changing Variable Types

In OMNIS Studio whenever you try to enter a variable with the same name as an existing variable of another type, the following message is displayed



where [Class] can be any of variable type.

If you answer “Yes” the original variable is removed, and all tokenized references to it are changed to reference the new variable. This occurs for all types of variable. If you answer “No” OMNIS Studio lets both variables coexist.

Generally you should avoid using the same name for variables of a different type.

Window Formats and Classes

When you convert a library in OMNIS Studio, all window formats are converted to window classes. All the window fields and objects in the old window format are converted and added to the new class, and all procedures are converted to methods. When you open a window class in OMNIS Studio you create an instance of the class.

Window Instances

In OMNIS Studio you can open multiple instances of the same window class providing they are given unique names. You use the instance name and not the class name to identify an open window with the notation or for most window commands. The *Open window* command has been renamed to *Open window instance* command. When you open a window using this command or the notation you can assign it an instance name. If you omit the instance name the class name is used by default, but the same name cannot be used twice. When you use the *Open window instance* command you can add /* after the class name to create an instance name on the fly, and repeated calls to the command *Open window instance MyWin/** will assign a unique name to each window instance. However you can open a window instance using the $open() method and return a reference to each instance created. For example, *Do MyWin.$open('*',kWindowCenter,param1,param2) Returns WinRef* will open the window *MyWin* in the center and return a reference to the instance, assuming *WinRef* is of type Item reference.

The *Open window maximized* and *Open window minimized* commands have been removed. In their place, the *Open window instance* command now takes a /MAX and /MIN parameter to maximize or minimize the window instance. Also the /STACK and /CENTER parameters have been shortened to /STK and /CEN.

The *Close window, Store window, Test for window open, Queue bring to top* and *Queue close* commands take a window instance name and not a class name as parameter. For compatibility you can specify a full class name, such as libname.windowname. In this case the instance name is taken as the simple name of the class.

The *Build open window list* command builds a list containing the names of the open window instances. The $name property of an open window now returns its instance name and not the class name.

The object group $winds has been renamed to $iwindows, and contains all the currently open window instances in the order in which they were opened. $cwind contains the name of the current window instance, and $windows contains all the window classes in your library.

Window Procedures and Methods

The procedures contained in an old window format are converted to methods in the converted window class, but they are separated into *Field methods* and *Class methods*. Each field can now contain multiple methods and the class itself has its own set of methods.

The converter copies each numbered procedure for existing fields and creates a new field method and removes the original procedure in the converted window class. *This means that any code which called them directly will now fail.* All the remaining methods not belonging to fields are placed in the class methods for the window class.

In OMNIS Studio you create and edit field and class methods in separate method editors. You can right-click on a field or window background to edit the field or class methods for the window class. As a shortcut, you can *double-click* on a field or window background to open the field or class methods for the window class. The method editor title bar for class methods is named "Window *ClassName* Class Methods" and for field methods it is named "Window *ClassName* Field Method *FieldName*".

In V3 the name of a procedure was taken from the field name or text for a pushbutton. When you changed the name or text of a field the procedure name was changed automatically. This functionality has been removed in OMNIS Studio, although the initial field name is still derived from the name of the field. Since methods are now identified by name it's not appropriate for OMNIS to change method names once they've been defined.

There will be some compatibility failures with notation which alters methods for a window. For example, notation such as

```
Do $cwind.$methods.xxx.$methodtext.$assign()
```

will now alter the window class method and not the field method, and

```
Do $cwind.$objs.xxx.$method.$methodtext.$assign()
```

will alter the field method and not the window class method.

Field and Button Procedures

Events for fields and other window objects are now controlled by a method called $event() contained in the field; this is called the *event handling method*. All procedures behind pushbuttons, button areas, and so on, are converted to $event() methods and placed in the Field methods for the field or window object. The converter adds an $event() method to most types of field, but it may not contain any code.

Initialization Procedures and $construct()

In OMNIS Studio classes that you can instantiate, including window, report, menu, and toolbar classes, can contain a $construct() method and a $destruct() method. These

methods handle the construction and destruction of the window instance. When you convert your library, the converter renames the window initialization procedure or Procedure 0 (line zero in procedures for the window) to $construct(). This method is executed automatically when you open the window instance. Consequently if elsewhere in your library the initialization procedure was called by name, this will fail after it is renamed $construct(). If your window initialization procedure contained a *Set window control procedure* command, this is retained for backward compatibilty but marked as *OBSOLETE* and continues to set up the specified method as the window control method containing your old event handling code.

There may be some compatibility problems due to $construct() being called directly as the instance is constructed unlike the V3 initialization procedure which was pushed on the stack. This may cause problems with libraries that make assumptions about the order and timing of events in your library.

Window Control Procedures

In V3 you could specify a particular procedure to be a control procedure for either a window format or a library using the commands *Set window control method* and *Set task control method* respectively. These commands are retained for backward compatibility in converted libraries, but are marked OBSOLETE and you should not use them in new libraries.

In OMNIS Studio events for a window field are handled in a method called $event() placed behind the field. Events for the window itself are also handled in a method called $event() that you add to the window class methods. Therefore the $event() method for a window is the equivalent of the V3 window control procedure. Windows can also have a $control() method which handle events passed to it from the $event() methods for fields in the window. Furthermore tasks can have a $control() method which handle events passed to it from window $control() or $event() methods.

For converted libraries only, a window control procedure set up using the *Set window control method OBSOLETE* command will continue to work, assuming $v3events is kTrue, and will override a $control() method in the window if there is one.

See the *OMNIS Programming* manual for a complete description of the new event handling system.

Window Field Names

In OMNIS Studio you should refer to a window field using its object name, that is, the name specified in its **name** property, and not its dataname, field number, or ident number. You should specify the data name of the field, that is, the variable or file class field name behind the field, in the **dataname** property for the field. To refer to a field on a window you can use its object **name**, windowname.name, or the full notation for a window field.

The windowname is the name of the window instance. A simple fieldname parameter is assumed to refer to a field on the topmost window instance, whereas the notation windowname.fieldname identifies a field on a particular window instance. Also you can use the full notation or square bracket notation to refer to any field or window instance.

This change affects a number of commands including the *Test for field visible* and *Test for field enabled* commands which now take a window field name. When you convert a library containing these commands the field numbers are converted correctly.

The *SNA set current field, Queue set current field* and *Send to window field* commands now take a window field name. These commands are converted correctly if they used a field number. However, if they specified a dataname or used indirection they will not be converted and are left unchanged as OBSOLETE commands.

The commands *Hide Fields, Show fields, Enable fields,* and *Disable fields* now take a window field name or list of field names instead of a field number or range of numbers. The old commands have been made OBSOLETE and replaced with equivalent commands that take a *field name* or list of field names as a parameter. These commands are converted correctly if they used a single field number, that is, the specified field number is converted to a window field name. However, if they specified a range of fields they will not be converted and are left unchanged as OBSOLETE commands.

Table fields

In V3 you could implement a group field using a table field with its rowsection turned off. Support for rowsection has been removed. Such group fields convert to the new Group field type containing the objects in the old table field.

Exceptions for table fields, now renamed grid fields, are implemented in a different way in OMNIS Studio. Existing grid field exceptions in converted libraries will no longer work. The new grid field exceptions are described in the *OMNIS Programming* manual in the *Window Programming* chapter.

Picture Fields

Block carets in picture fields have been replaced by a dotted line. When a picture field gets the focus it displays a dotted line around its border.

Window Redrawing

The window redrawing mechanism has changed in OMNIS Studio. The appearance of an open window can change when you use notation or when you use a *Redraw* command to show the current value of a field or variable. Such changes in appearance were queued in V3 until the window was refreshed, that is, V3 waited for user input or for a repeat, while

or for loop to be executed. Consequently, some windows were not refreshed when required or they were refreshed too often while executing a loop.

In OMNIS Studio the following changes have been made that affect window redrawing.

- A window is not redrawn when you change the value contained in a field's dataname
- Refreshes no longer happen at While, Repeat or For loops
- The old *Redraw* commands are replaced by a single *Redraw* command
- a $redraw() method has been added for window fields, open windows, and $root
- all windows currently open are always refreshed when OMNIS stops and waits for user input, including at *Enter data* and when an OK or Yes/No message is displayed

$redraw() method

Window fields, open windows, and $root now contain a $redraw(*setcontents*,*refresh*) method. $root.$redraw() redraws all window instances currently open. The *setcontents* parameter causes the contents of the field, window or all windows to be reset, and the *refresh* parameter causes the window or windows to be refreshed. If you omit the *setcontents* and *refresh* parameters, OMNIS defaults to $redraw(kTrue,kFalse), which means the contents of the field is reset, but the change would not be visible.

Redraw commands

The various *Redraw* commands in V3 are replaced by a single *Redraw* command which accepts a list of fields or window names. The list of names can contain simple field or window names or names prefixed with a library name. The *Redraw* command resets the contents of the specified fields or windows, and if you check the **Refresh now** option the window or windows containing those fields are redrawn immediately.

The *Redraw windows*, *Redraw named fields*, and *Redraw numbered fields* commands are marked as OBSOLETE and you should not use them in new libraries. The *Redraw lists* and *Redraw menus* commands are unchanged.

Calculate command and Redraw field

The *Redraw field* option has been removed from the *Calculate* command. When you convert a library that uses this option the converter will insert a comment after the *Calculate* line. You should replace this comment with the appropriate *Redraw* command.

Toolgroups and Toolbars

In V3 toolbar controls were stored as toolgroups within a window or menu format. In OMNIS Studio toolgroups have been removed. Groups of toolbar controls are stored as separate toolbar classes.

The converter removes each toolgroup from window or menu formats and converts them to individual toolbar classes called T_OLDWINDOWNAME or T_OLDMENUNAME, where OLD..NAME is the name of the converted window or menu format. Existing references to the old V3 toolgroups are changed to reference the new toolbar classes. The method behind each control in the new toolbar class calls the corresponding method in the old window or menu format. For example, a button in a converted toolbar class will contain an $event() method containing a *Call method* to the appropriate method in a window or menu class.

Events

OMNIS Studio introduces a completely new event handling mechanism that supersedes the V3 events mechanism. In OMNIS Studio an event is reported as a series of parameters to the event handling methods contained in an object. For example, a click on a field is reported as an evClick event and is sent to the $event() method behind the field. Event handling methods behind the objects in your library can contain the *On* command plus an event code to detect and handle specific events. For a complete description of the new event handling system, see the *OMNIS Programming* manual.

There is a new library preference called $v3events that enables or disables the old events system. In converted libraries $v3events is kTrue by default, which means your libraries will function as before. For example, in converted libraries the SNA commands and old message variables such as #BEFORE and #AFTER continue to work.

If you want to make sure v3 events are enabled in your converted library, you can insert the following line in the $contruct() method of the Startup task

```
Do $clib.$prefs.$v3events.$assign(kTrue)
```

If you create a new library in OMNIS Studio, $v3events is turned off and the old events system and message variables will not work.

The message variables, such as #CLICK, #AFTER, #MOUSEENTER, will work in converted libraries, but they have been removed from the Catalog and you should not use them in new libraries.

Note that the global hash variables such as #L1, #S1, #1..#60 still appear in the Catalog and continue to work as before, but you should avoid using them in new libraries. For

example, instead of using #L1 you can declare a list variable in a task, class, instance, or method to give you a list of the required scope.

Control Procedures and Events

In OMNIS Studio events for a window field are handled in a method called $event() placed behind the field. Events for the window itself are also handled in a method called $event() that you add to the window class methods. For example, events such as evToTop are handled in the $event() method for the window. The $event() method for a window is the equivalent of the V3 window control procedure. Windows can also have a $control() method which handle events passed to it from the $event() methods for fields and objects in the window. Furthermore tasks can have a $control() method which handle events passed to it from window $control() or $event() methods.

For compatibility the *Set window* and *Set task control method OBSOLETE* commands continue to work in converted libraries and will override their corresponding $control() methods if present. You can comment out the old *Set window* and *Set task control method OBSOLETE* commands and enter your own $control() method for a window or task class.

Field Events

In V3 field events were sent to the procedure behind the field, then to the window control procedure, and then to the library control procedure. In OMNIS Studio events for a window field are sent to the field method called $event(), to the window $control() method, and to the task $control() method. The $event() and $control() methods can contain On commands to handle specific events at the appropriate level.

In OMNIS Studio, a control method is called only if the field that received the original event doesn't contain a handler for that event, or if the field event method explicitly passes control to the window control method. You can use the *Quit event handler* command with the *Pass to next handler* option to pass an event up the event chain.

The new event mechanism contains only one event code evBefore to detect when the cursor enters a field; you cannot distinguish #BEFORE1 and #BEFORE2 as in V3.

Window Events

Window classes can contain a method called $event() which is used to detect events in the window, such as clicks on the window background. Window events are sent to the $event() method and then to the task $control() method. To help avoid compatibility failures, if $v3events is on and there is no $event() method in the window class, window events are sent to the window $control() method.

#TOTOP and #BEFORE are now simple notification messages so #ER values are no longer sent with these.

The new event mechanism contains only one event code evToTop to detect when a window comes to the top; you cannot distinguish #TOTOP1 and #TOTOP2 as in V3.

Menu Events

In V3 some events associated with menus were sent to the library control procedure if no user windows were open. These events are no longer sent. Each menu line contains an $event() method which is used to detect events for the menu line, that is, $event() is called when the menu line is selected.

Enter Data Events

The disabled field events with #EM=6, #EM=7 or #ER>4000 are no longer reported and have no equivalent new event code. However #ER>3000 is still reported as an equivalent to evMouseDown on a disabled field. When a new event has no V3 equivalent #EM is sent as zero.

The #ER values in the range 5003 to 5009 are no longer reported for the OMNIS database buttons (Next, Previous, and so on), instead these generate an evStandardMenu event, which causes #ER to be in the range 16002 to 16008.

OMNIS Database Events

In converted libraries, the database events, such as #NEXT and #PREVIOUS, still work as before provided the $v3events preference is turned on.

SNA Commands

In converted libraries when $v3events is on you can use the SNA commands for events which support SNA in V3. However for new types of events SNA is not supported. You should avoid using the SNA commands in your event handling methods. To simulate their functionality, you should use the *Quit event handler(Discard event)* command to discard an event and perform whatever next action is required.

To assist with compatibility, when you use an SNA command in an event handler the current event is discarded (with the exception of *SNA perform default action*). This rule applies only for those events which support SNA in V3 and continue to support it in OMNIS Studio. For all new types of event the SNA commands are ignored.

Popup Menus and Events

In OMNIS 7 Version 3.5, #CLICK was called on the button down event rather than the mouse up event, as in older V3 versions. This ensured that the *Popup menu* command worked inside a window control procedure, especially under MacOS. In OMNIS Studio #CLICK and evClick, its replacement, both occur on the mouse up event. Therefore popup

menus implemented in V3.5 no longer work in OMNIS Studio. However in OMNIS Studio you should use *context menus* to implement menus that poup over windows or window objects.

List Variables

In V3 each list column referenced a field or variable name, and commands such as *Load from list* copied the column values into their field or variable names. These variable names are held in the $dataname for the column, which is the equivalent of the V3 $fieldname. Lists defined in V3 still have datanames, but lists with a class or columns defined using $cols.$add(column_name,type,subtype,length) no longer have datanames.

If you use *Load from list, Replace line in list,* and so on, for a list without datanames it is necessary to use parameters with these commands to denote where the values are to go to or come from. For example

- **list_name.column_name**
  refers to the data item stored in the specified column for the current row. This is standard notation very much like ‘table_name.column_name’.
- **list_name.row_number.column_name**
  can be used instead of list(“column_name”,row_number) to refer to a particular data item in a list and avoids having to quote the column name.

The different meanings of “list_name.column_name” and “list_name.row_number” are potentially ambiguous and OMNIS Studio resolves the ambiguity by taking character values to be column names and everything else to be row numbers. This is mainly a problem with square bracket notation. Therefore, if a character variable such as CHAR1 contains a row number you must use “LIST.[CHAR1+0]” and not “LIST.[CHAR1]”.

The items “list_name.column_name” and “list_name.row_number.column_name” are notation so, in common with all notation, if the list is a file class field you must use “file_name.field_name” to refer to the list and not “field_name”. This is irrespective of the state of the $uniquefieldnames library preference.

Storing item references in a list

OMNIS Studio lets you store item references in a list. This feature introduces some minor incompatibilities, which apply to both list and row variables.

If an item reference is specified as a list column using the *Define list* or *Define list from table* commands, or the $define() or $definefromtable() methods, the column type for that column becomes item reference. The values stored in, or extracted from, item reference columns are obtained using *Set reference*, rather than using the $assign() method. This applies to:

- adding lines to a list using *Add line to list, Insert line in list, Replace line in list* or $assigncols()
- loading a value from a list using *Load line from list* or $loadcols()
- $makelist() if a column which evaluates to an item reference is assigned
- $appendlist() and $insertlist() when loading the column values for columns of type item reference

Individual list elements which belong to a column of type item reference act in the same way as an item reference variable. So you use the *Set reference* command to set up the item reference (if not already set up by *Add line to list*, $assigncols(), and so on) and normal usage refers to the referenced item. Therefore

```
Calculate FRED as 'OBJECT1'
Set reference MyRef to $cinst.$objs.FIELD1
Add line to list LIST (FRED,MyRef)
```

Report Formats and Classes

In OMNIS Studio when you print a report class you create an instance of the report. Report instances contain a $objs and $sorts group containing the objects and sort fields for the report instance. You can use $sorts to determine when subtotal and page breaks will occur, although the report instance itself does not carry out the sorting. You cannot use the $add() and $remove() methods on a report instance since the report has started printing.

Measurement systems for a report instance are slightly different than for a report class. Measurements for the instance are held in output device coordinates and then rounded to inches or cms.

Report Totals

In V3 the capability existed to include a field aggregate value in a calculation or square bracket notation in a subtotal or total section by simply using the name of the field. A limitation of the V3 implementation was that each field could have only one type of aggregate calculated.

In OMNIS Studio each report field contains the methods $total(), $average(), $minimum(), $maximum(), and $count(). To use these aggregate methods, you specify the dataname of the field followed by the method name. For example, myField.$total() provides the running subtotal and total values for the field, and myField.$average() provides the average for the field. These methods only work for subtotals and totals sections, and replace the need to use a calculation or square bracket notation.

Record and Page Count Variables

The property $reccount has been added replacing #R. It contains the number of record sections printed. You can use $reccount.$total() in a subtotal section to obtain the number of records included in that set of subtotals.

Also $pagecount has been added replacing #P. It contains the page number of the current printing position, and is the same as $cposition.$page.

In addition $subvariable has been added replacing #SUBFLD. It holds the name of the subtotal variable when a subtotal or subtotal header section is being printed.

Export Formats

The report export formats kEXdif, kEXsylk, kEXdBase, and kEXlotus have been marked as OBSOLETE and you should not use them in new libraries. The $exportformat property now takes kEXcommas, kEXtabs, kEXopl, and kEXodt only. Note that you cannot change the export format once the report has started printing.

Report Fields

The Data Field Name of a report field is now the dataname and the corresponding notation is changed from $fieldname to $dataname. For a report field, $dataname is the variable or file class field name of the report field. For a report sort field, $dataname returns the name of the variable being sorted.

Help Messages

The *Help message* command has been removed. However, window fields contain the **tooltip** property and menu lines contain the **helptext** property in which you can enter short descriptive messages. When you convert a library, help messages behind window fields are transferred to the **tooltip** property for each window object, and messages contained in menu formats are converted to the **helptext** property for the new menu class and each menu line.

Icons

The icon data file supplied with OMNIS Studio, called OMNISPIC.df1, contains the icons used throughout the OMNIS Studio development environment and in your own window fields and toolbar controls. Each icon in the icon data file has a unique ID. When you want to add an icon to an object you set its $iconid property.

OMNIS Studio provides an icon editor that lets you create your own icons for picture buttons, toolbar controls, and so on. You should not edit OMNISPIC.df1 or add your own icons to this data file, rather you should add your own icons to a separate data file called USERPIC.df1. If you edit the icons or change the IDs in OMNISPIC.df1 the correct icons will not appear in OMNIS Studio or your libraries.

The OMNISPIC.df1 data file supplied with OMNIS Studio has a completely different format from the one supplied with OMNIS 7 Version 3. Therefore you cannot use the old OMNISPIC.df1 data file in OMNIS Studio unless you convert it to the new format.

The icon data file called USERPIC.df1 supplied in the Icons folder is the old OMNISPIC.df1 file converted to the new format. This ensures that your converted libraries have the correct icons for all picture buttons and other controls. However following conversion some objects in your library may have the wrong icons since the OMNISPIC data file gets priority over USERPIC in ID conflicts.

If you haven't changed the old OMNISPIC.df1 file at all you don't need to do anything, that is, ignore the following section. However if you've changed your old OMNISPIC.df1 file or added icons to it, you will need to convert it and replace the USERPIC.df1 data file supplied in OMNIS Studio. Before you convert your old OMNISPIC.df1 file, make sure you have a secure backup.

To convert your old OMNISPIC data file

- Quit OMNIS Studio
- Remove or rename the USERPIC.df1 file supplied with OMNIS Studio in the Icons folder
- Restart OMNIS Studio
- Select the Tools>>Icon Editor option on the main menu bar
- Select the File>>Convert Old OMNISPIC option in the Icon Editor
- Find and select your old OMNISPIC.df1 file

- Go to the Icons folder in the main OMNIS folder and save the converted file as USERPIC.df1
- Restart OMNIS Studio

When you restart OMNIS Studio the USERPIC.df1 file will be loaded automatically. If you want to add icons to OMNIS Studio you should add them to the USERPIC.df1 data file or #ICONS in your library, as described in the *Using OMNIS Studio* manual.

OMNIS icon data files now support full page icons in which you specify their size.

Colors

In V3 the color of an object was stored as a number between 1 and 272 corresponding to a color in the OMNIS color system table. The #COLORS system table and all associated notation has been removed. Colors are now represented using RGB values, therefore an object stores its color as an RGB value. All code in your old library that tries to set the color index of an object will now fail, and the color of some objects will be different in your converted library.

In OMNIS Studio you can use the color constants, which represent specific RGB values, to set the color of an object. For example, to make an object yellow you can use the kYellow constant, or set its RGB value to 255,255,0. In OMNIS Studio an object cannot be transparent, that is, it cannot be set to have no color; most window objects default to kDefaultColor, kColor3dFace, or kColorWindow which ensures the objects have the correct color, particularly under different operating systems.

Graphs

External areas have been removed in OMNIS Studio, and are replaced by external components. Consequently any graphs in your old library will be removed. The converter cannot convert the graphs in your library, therefore you will have to replace them with graph external components available in the Component Store.

Ad hoc Reports

The V3 ad hoc report engine has been removed from the core product and replaced by a separate library which you can load and use like any other library. The new ad hoc report library will convert your old ad hoc report files, although some of the formatting will be lost. Fields in ad hoc reports now use the $formatmode and $formatstring properties, and the converter will set these properties for you depending on the type of field.

Note that ad hoc reports are stored as library files, with the .AHR file extension, therefore when you open an old ad hoc report file OMNIS will prompt you to convert the library.

DAMs and External Names

All Data Access Modules, or DAMs have been renamed in OMNIS Studio; their names now begin with the letter “D”. For example, the Oracle DAM is now called Doracle.dll under Windows, or dORACLE under MacOS. When you convert a library all DAM names are changed automatically, but you may need to change any references in strings or elsewhere in your library. For example, the *Start session* command converts to the new names. All DAMs are found in the EXTERNAL folder.

Likewise all externals, including extended command and function packages, have been renamed in OMNIS Studio; their names now begin with the letter “X”. OMNIS Studio loads certain externals automatically so you don’t need to make any changes, but you may need to change any references to externals in your own libraries. If an external is missing or cannot be found references to it will fail causing a runtime error.

Commands Removed by Converter

When you convert your library the converter will remove certain commands without replacement. See the *What’s been removed* section for a complete list of commands that have been removed.

The converter will comment out any commands that have been removed in OMNIS Studio. For example the commands *Other parameters are optional*, and *Clear class variables when closed* have been removed and will now appear in your code like this

```
; Other parameters are optional       ;; COMMAND REMOVED BY CONVERTER
; Clear class variables when closed   ;; COMMAND REMOVED BY CONVERTER
```

Checking Methods in a Converted Library

You can check the methods in a converted library using the *method checker*. The method checker is available under the Tools menu on the main OMNIS menu bar. It can check the code in a converted library for syntax errors, unused variables, methods without code, and commands that are removed by the converter.

Note that the method checker does not correct the code in a converted library automatically, it simply reports any errors and potential problems in your code.

You must open your library in OMNIS Studio to convert the library *before* opening it in the method checker. When you open the method checker it will load your converted library, and any other libraries you have open.

To check the methods in a converted library

- Select the Tools>>Method Checker menu item from the main OMNIS menu bar
- Double-click on your converted library to open the Select Classes dialog
- Select the classes in your converted library that you want to check, or click on the Select all classes button
- Click on the Check button to start checking

The method checker works through the classes you selected displaying their statistics in the Method Checker Error Log. You can cancel checking at any time by pressing Ctrl-Break/Cmnd-period.

When checking is complete, you can sort the log by clicking on one of the headers in the list. You can print the log or save it to a text file. You can show a summary of the errors for each class by clicking on the Show Summary button.

The method checker searches for many different problems, but for converted libraries the following error messages may apply. For a complete list of error messages, see the *OMNIS Programming* manual.

Missing extended command or function
A missing extended command or function was encountered, either not loaded or installed: these show in your code beginning with the letter “X”. Your converted library may contain a reference to an extended command or function that has been removed in OMNIS Studio.

Class variable with the same name as a task variable
Could cause precedence problems at the class level.

Named method with no code
Check to see if this method is required. The converter removes certain commands no longer supported in OMNIS Studio leaving empty methods. OMNIS Studio also adds a $construct() method to each class which at this stage may contain no code.

Obsolete command
You should not use obsolete commands: remove them from your code. For example, you can replace *Call method* with *Do method* or *Do code method*.

Command removed by converter
In converted libraries certain commands are commented out: you may need to enter an alternative command.

Reference to hash variable
Avoid using hash variables: replace with variable of appropriate scope.

Index

OMNIS

OMNIS Studio Conversion

Version 2

How to use this manual

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

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

Bookmarks

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

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

Thumbnails

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

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

Links

Links in this document connect related information or take you to a specific location in the document. Links are indicated with *blue italic* text. To jump to a related topic, move the pointer over a linked area (the pointer changes to a pointing finger) and simply click your mouse. Try it!

To return to your last view or location, click on the **Go back** button on the Acrobat toolbar.

Browsing

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

Find

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

Search

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

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

Grabbing Text from the Screen

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

Getting Help

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