Using OMNIS Studio

OMNIS Software

August 1998

Table of Contents

About This Manual

This manual describes how you develop an application using OMNIS Studio, and focuses on the classes, objects, and tools available in OMNIS Studio. The *OMNIS Programming* manual covers more advanced topics, including application design, programming, and platform specific features. The *Using* and *Programming* manuals are also available on the OMNIS CD in Acrobat PDF format, together with the following manuals:

– *OMNIS Studio Conversion*
  describes how you convert your OMNIS 7 applications to OMNIS Studio and, for the benefit of OMNIS 7 users, introduces the new features in OMNIS Studio

– *OMNIS Graphs*
  describes the Graph external component supplied with OMNIS

In addition to these manuals, a comprehensive Help system describing the OMNIS Commands, Functions, Properties, and Methods is available from within the OMNIS development environment under the Help menu or by pressing F1.

Your Notes

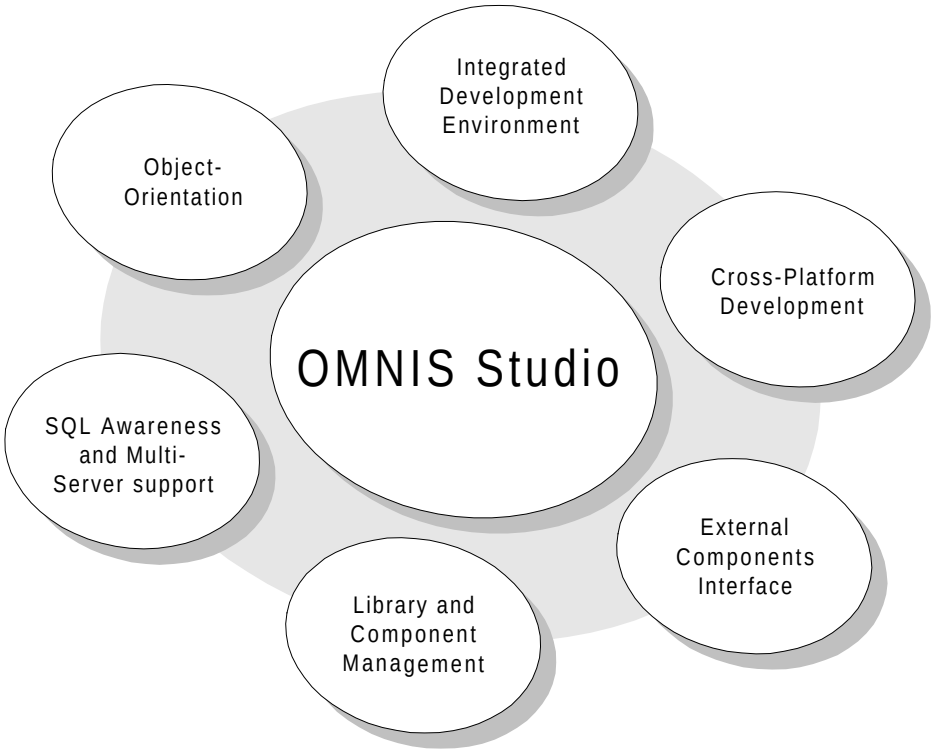
Chapter 1—Introduction

This chapter presents an overview of OMNIS Studio and describes the objects and components you need to create an OMNIS application.

What is OMNIS Studio?

OMNIS Studio is a rapid application development (RAD) environment that supports component-based development and integration as well as providing access to a full range of databases and Internet services.

OMNIS Studio provides the tools for complete *library* and *component management*, and offers *cross-platform* development and deployment under Windows and MacOS. You can extend its capabilities by adding third-party components and development tools, such as CASE and GUI testing tools.



The OMNIS Studio Data Access Manager provides *multi-server support* via its *Data Access Modules* or DAMs, and automatic client-to-server data mapping using *SQL-aware* classes.

OMNIS also incorporates many familiar *object-oriented* features including class reuse, inheritance, messaging, and data encapsulation that enhance development.

OMNIS Applications

With OMNIS Studio you can create applications to perform almost any task, including payroll and expense tracking, human resources and administration, healthcare, electronic banking and commerce. To build an application you need to piece together various building blocks or components, including

- the OMNIS executable or program
- one or more library files
- DAMs for server database access
- OMNIS data files for local data access
- externals, DLLs, and external components

The OMNIS Executable

The OMNIS *executable* or program file directs the objects in your application and provides the Integrated Development Environment, or IDE, in which you design your application. It also provides the *runtime* environment in which the user runs your finished application. Many of the features available to you as the designer to create, debug, and modify your application are invisible to the user at runtime.

OMNIS Libraries

The main component in your application is the *library file*. This file contains all the class definitions that define the data and GUI objects in your application. The class types provided in OMNIS include window, menu, report, and toolbar classes.

If you need to access data on a server database, you use OMNIS Software's OMNIS Studio *Data Access Modules* (DAMs) which let you connect to all the leading relational DBMSs including Oracle, Sybase, Informix, and Microsoft SQL Server. Alternatively, if you need to store data on your local machine or LAN, you can use the OMNIS relational database, either directly or using SQL. OMNIS databases are held in files called *data files* and OMNIS provides a number of tools that let you create, browse, and modify data files.

OMNIS Object Orientation

OMNIS Studio uses many of the concepts and terms from Object-Oriented Programming, including messaging, inheritance, and code reuse. This section introduces some of these concepts and describes how you can use them to develop your application using OMNIS.

Libraries and Classes

Developing an OMNIS application is all about creating the right objects and changing how these objects interact with each other and the end user. An OMNIS *library* is a complex object that can contain many different types of data and GUI objects and components including windows, menus, toolbars, and reports. The definition for each of these objects is stored in your library as a separate *class*.

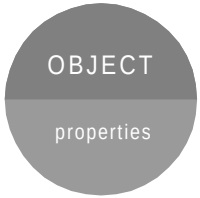
OMNIS Studio provides several tools that let you create and examine the libraries and classes in your application. The main tools that let you do this are the *Browser* and the *Component Store*.

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 fields and controls, report objects, and external components.

When you start OMNIS the Browser and Component Store open automatically, ready for you to start creating libraries and classes.

Properties

Every object has certain characteristics that define exactly how it looks and behaves. These are called its *properties*. Properties of an object might include its name, type, color, size, border style, visibility, and so on.



Every library has its own set of properties, classes have their own properties, and individual objects such as fields on a window have properties as well. Some properties are common to all objects, such as name, and some are unique to a specific type of object.

You can examine and change the properties of any object in OMNIS using the *Property Manager*. For example, you can view the properties of your library or a particular class using the Property Manager. It lists all the properties for the currently selected object and you can open it from almost anywhere in OMNIS.

Methods

Objects contain *methods* which are pieces of code that perform some action when you send the object the appropriate message. For example, window classes contain a method to *construct* the class. In this case, when you open the window the construct method is executed which initializes the window.

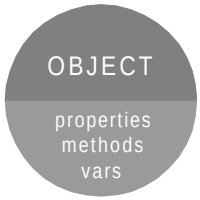
Objects contain default methods, but you can add your own methods that either add to the object's functionality or override its standard behavior. You use the *method editor* to add methods to an object.

Your own methods can contain OMNIS *commands* that define exactly what the method should do. OMNIS has over 400 commands that do everything from opening a window, installing a menu, printing a report, to responding to events in your application.

You can add methods to most types of class and the objects within the class. For example, you can add methods to a window and to each of the objects on the window, and you can add methods to a toolbar class and put methods behind each of its buttons and controls.

Variables

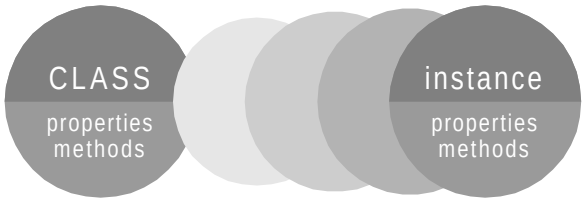
The principal data container in OMNIS is the *variable*. Most objects can contain variables, but their scope and the kind of data they can contain depends on the variable type.



You can add variables to most types of class. Such *class variables* are visible only within that class, and their values are available to all the methods in the class. Also you can add variables to any method. These are called *local variables*, and are available only to the method in which they are defined. You add variables to an object using the *method editor*.

Instances

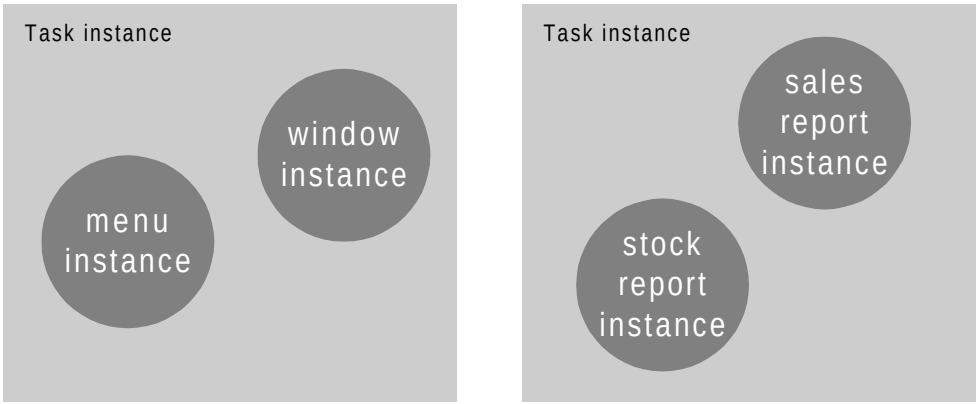
An *instance* is the object you create when you *open* a class. For example, you create an instance of a window when you open a window class. When you print a report you create an instance of a report class, and similarly when you install a menu you create an instance of a menu class.



When you open an instance of a class it inherits all the properties, methods, and variables from the class. In addition to class variables, each 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. Instances of any class must always be opened in a task. To ensure this, OMNIS provides a *startup task* for instances to run in, or you can create your own task. Since instances are owned by the task they are running in, manipulating the task manipulates all instances owned by the task. In this way you can hide and show all the instances or close all the current instances belonging to a particular task.



Notation

OMNIS structures its objects in an object tree, or hierarchical arrangement of objects and groups that contain other objects. To facilitate a system of naming or referring to the objects in your library, and their properties and methods, OMNIS uses a system called the *notation*. You can view the OMNIS object tree and the notation using the *Notation Inspector*.

In the OMNIS notation all property names start with a dollar sign “$” to distinguish them from things like the name of your library or class names. Properties are single words containing only letters or digits. For example, the name of an object is $name, the width of an object is $width, its text alignment is $align, and so on.

Standard method names also begin with the dollar sign, but are further distinguished from properties by having parentheses after their name. Therefore the open method for a window is $open(). You also use the notation to execute a method or to change the properties of an object, and you can use a notation string anywhere you need to reference a variable or field name.

As well as properties and methods, the notation includes standard objects and groups that also start with a dollar sign. Therefore, the top level of the object tree is $root which contains everything, including your library and its contents. All the menu classes in your library are stored in the $menus group, all the report classes are contained in the $reports

group, all the toolbars are in the $toolbars group, and so on. In a similar way, the objects created at runtime when you run your application are contained in their own groups: hence the $imenus and $itoolbars groups contain all the installed menus and toolbars, and $ireports contains all the current report instances.

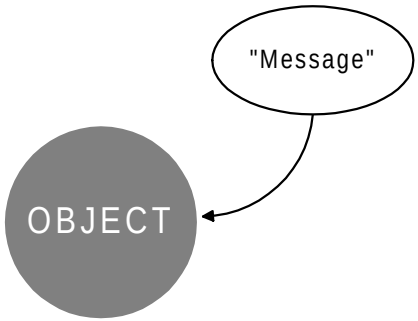
To write the full notation for an object you need to include each object and group in the object tree, separating each object using "." a dot. If you like, you can consider the notation for an object as the path to the object within the system of objects in your library, or OMNIS as a whole. For example, to refer to a window class in your library you would use the expression

```
$root.$libs.Libraryname.$windows.Windowname
```

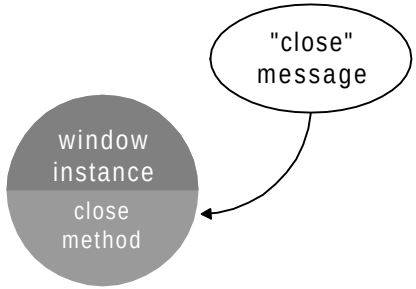
If you want to refer to a particular object on your window you need to add the $objs group and the name of the object to this string. Note that libraries are contained in a group called $libs, windows in one called $windows, and the objects on a window in a group called $objs, and you need to include these in the notation for an object on a window. However, when you write notation there are a number of shortcuts that let you reference objects without always referring right back to the $root object; these are discussed in the *Programming Methods* chapter later in this manual.

Messages

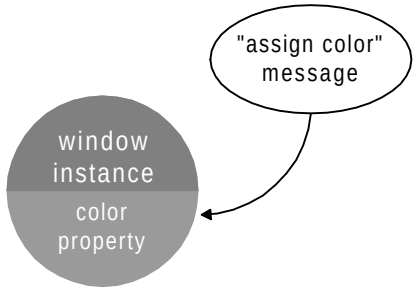
The essential components of any Object Oriented Programming system are the objects in that system and the *messages* sent to and from those objects. OMNIS is no exception.



A *message* is an instruction to do something to an object, such as open, close, print, or redraw the object. 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 manner to the notation, where each message begins with a "$", and then the name of the message. For example, you can send a $close() message to a window instance which runs the standard *$close() method* for that instance, which will close the 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.

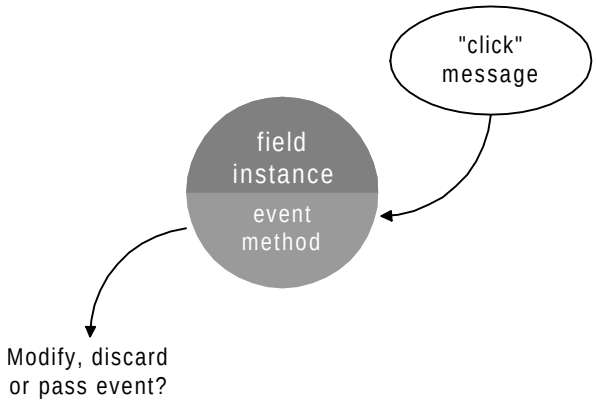


As well as assigning *to* an object, you can return information *from* an object using messages. For example, you could find out the background color of a window instance by returning the value of its color property. You do this using the notation as well.

Events

Any user action that occurs in OMNIS 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 put them behind fields and windows, or place them in the tasks in your library.

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.

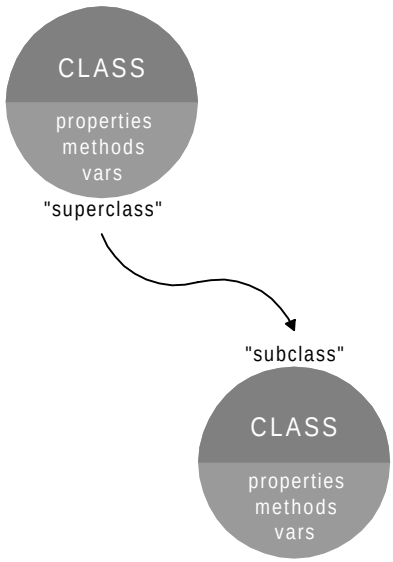


The event handling method can modify the default action for the event or redirect method execution to anywhere else in your application. Alternatively, the event handler may discard the event altogether or pass it on to the next method in the event chain.

Most objects including window fields and classes, toolbar controls, and menu lines contain a default event handling method called $event(), while window classes and tasks can contain a method called $control() as a further level of control. These methods contain the code that handles events passed to the object.

Inheritance

When creating a new class, instead of having to set up all its properties and methods 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.



So what are the advantages of inheritance? The real advantage of inheritance is that a subclass takes on or *inherits* all the properties, variables and methods of its superclass. And in the case of a window, menu and toolbar subclass, it inherits all the fields and objects on the superclass as well. This saves time and effort when you develop your application, since you can reuse the objects from the superclass. When you make a change in the superclass, all its subclasses inherit the change automatically.

In addition to its own inherited properties and methods, you can add methods and any other objects to the subclass.

You can view the inheritance structure for all the classes in your library using the *Inheritance Tree*.

Chapter 2—OMNIS Tools

This chapter shows you how to start OMNIS, and introduces the development tools and standard menus available in OMNIS Studio. The following tools are described in this chapter.

- *Context and View menus*
  general purpose tools that greatly accelerate development
- *Browser*
  for creating and viewing libraries and classes
- *Component Store*
  for creating classes and other library objects
- *Property Manager*
  for examining and changing the properties of objects in your library
- *Notation Inspector*
  for examining the OMNIS object tree including the contents of your libraries, and getting the correct notation for an object
- *Inheritance Tree*
  for examining the superclass/subclass structure of your libraries
- *Method Editor*
  for adding variables and methods to classes and objects; also for debugging your application
- *Interface Manager*
  for displaying public methods for any object in OMNIS Studio
- *Catalog*
  lists all the variables in your library and provides a convenient list of all OMNIS functions and constants
- *Data File Browser*
  for viewing and managing OMNIS data files
- *Standard menus and toolbars*
  for accessing the IDE and your libraries

Starting OMNIS Studio

Starting OMNIS Studio is easy! Assuming you have carefully followed the installation instructions provided with the product, you can start OMNIS Studio immediately.

To start OMNIS Studio under Windows 95 and Windows NT 4.0 or later

- Click on the Windows Start button and select the OMNIS Studio menu item

or

- Open Windows Explorer and locate the OMNIS folder
- Double-click on the OMNIS executable icon

To start OMNIS Studio under Windows NT 3.5.1, or Windows 3.1

- Open Program Manager and locate the OMNIS program group
- Double-click on the OMNIS program icon

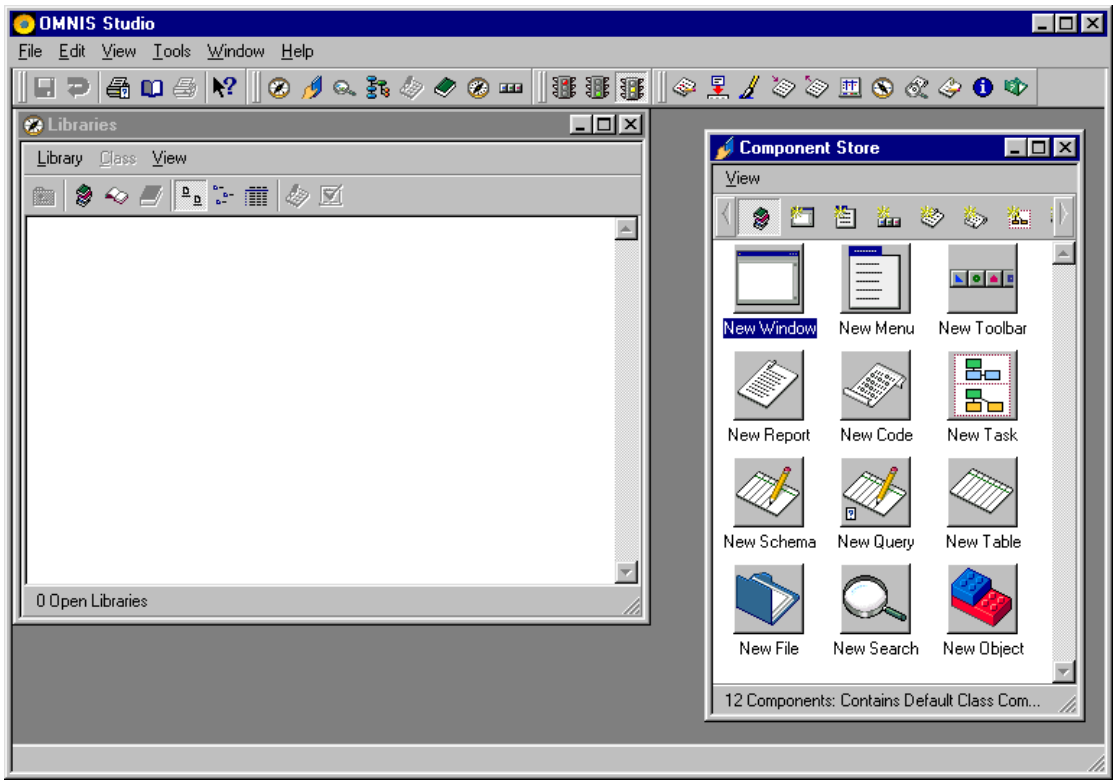
or

- Open File Manager and locate the OMNIS directory
- Double-click on the OMNIS executable

To load OMNIS Studio under MacOS

- Locate the OMNIS folder and double-click on the OMNIS program icon

When you start OMNIS Studio the main menus and IDE toolbars are installed, and the Browser and Component Store are opened. You will use all these tools to create and manage libraries and other components in your application.

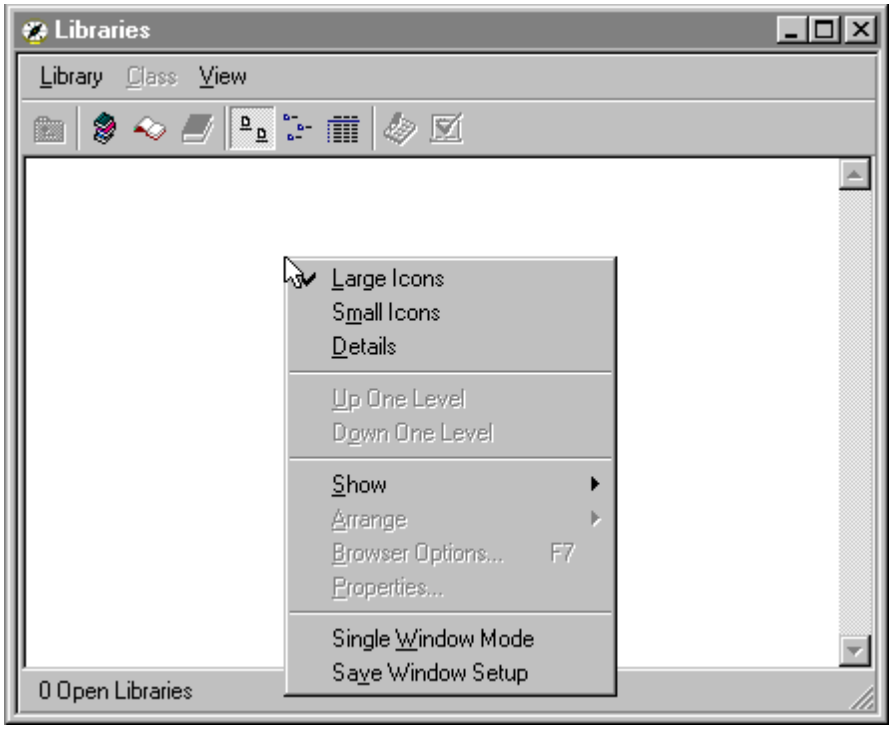


When you start OMNIS Studio for the first time the Welcome to OMNIS Studio library is opened. From this you can take a tour of the features in OMNIS Studio, run a brief tutorial, access the Examples Browser, or create a new library. You may like to take the tour and/or try the tutorial, otherwise you can create a new library or click on the Cancel button to close the Welcome library. You can choose not to see the Welcome library the next time you start OMNIS Studio. If the Welcome library opens and you have already created a library, a further option lets you open your most recently used library.

Note that you can create your own Welcome library to introduce your own application; this is described in the *Library Tools* chapter.

Context and View Menus

A *context menu* is a useful shortcut when you want to modify an object, or change the OMNIS development environment. You can open context menus from almost anywhere in OMNIS by Right-clicking on the object or window, including most of the browsers and tools described in this chapter; 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 key and click the mouse under MacOS. The options in a context menu will vary depending on the tool or object you click on, but you will always get the options you need for the current context. For example, you can Right-click on the Browser to open its View menu.



Browser showing the View context menu

View menus let you change the current view or behavior of a tool. Most of the design tools in OMNIS have a View menu on their menu bar, including the Browser and Component Store. You can also Right-click on the background of a tool to open its View menu. When you change how a tool is displayed you can save its setup using the Save Window Setup option in the context or View menu for the tool.

Several of the tools let you display *help tips* or *tooltips* from their context or View menus. These are short descriptive messages for the current object that are displayed when you position the mouse over an object or toolbar control.

Browser

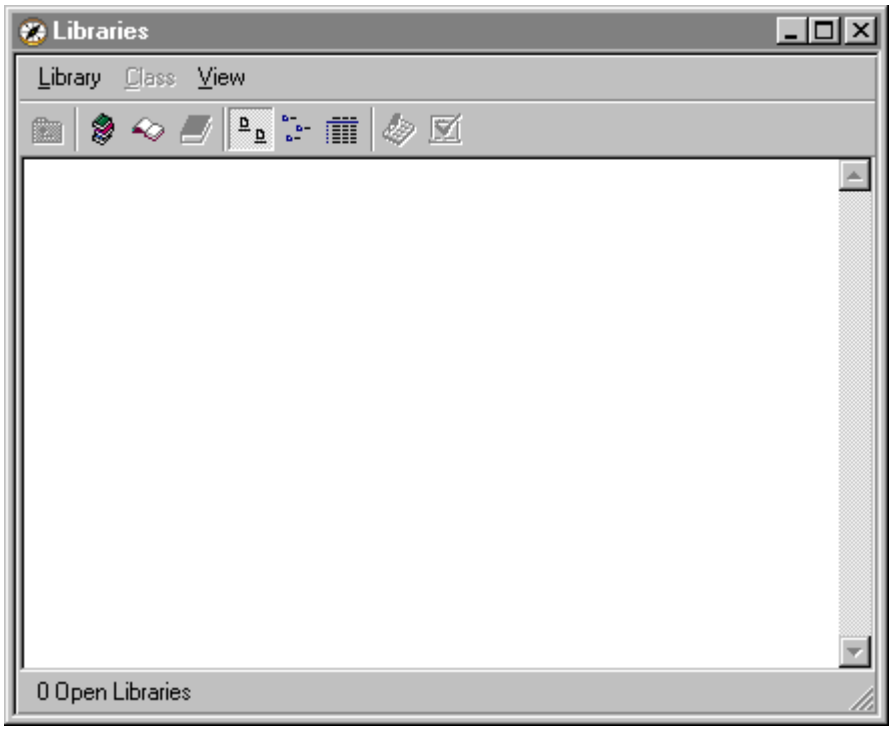
The Browser opens when you start OMNIS. It lets you create new libraries and add new classes to the current library. It displays all your open libraries or all classes in a particular library. You can open the Browser from the View menu, or by pressing F2/Cmnd-2.

To open the Browser

- Select the View>>Browser option on the main menu bar

or you can

- Press F2/Cmnd-2



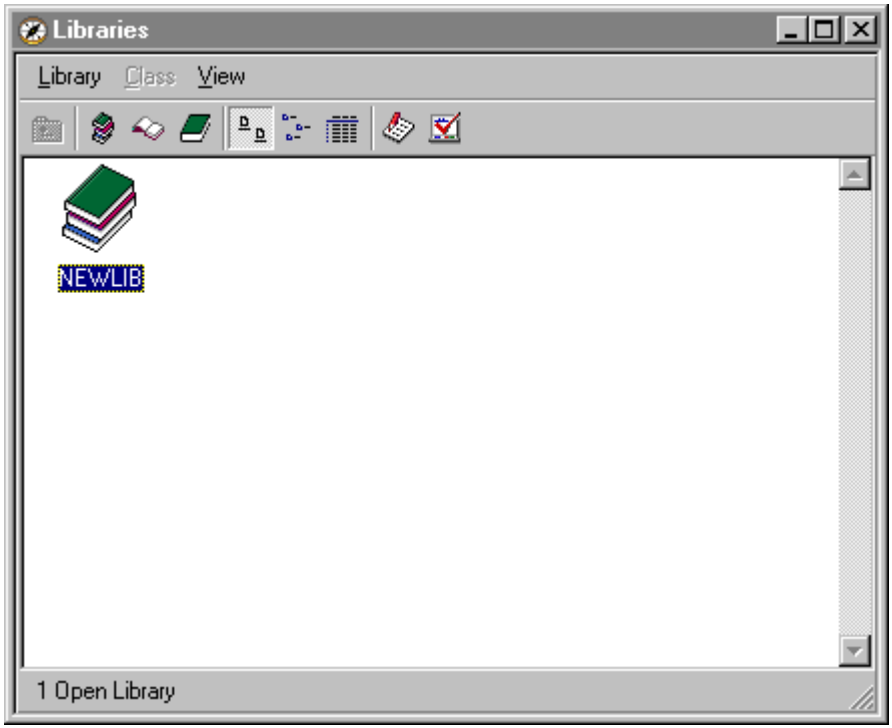
When you start OMNIS the Browser will be empty. At this stage you can either create a new library or open an existing library. Note that you can also create a new library from the

Welcome library, but this section assumes you have finished with and closed the Welcome library.

To create a new library

- Select the Library>>New menu option on the Browser menu bar
- Enter a name for the library, including the file extension .LBS, and click on OK

When you name a new library the .LBS file extension is not obligatory but it will help you distinguish library files from other types of files. Note that the Browser does not display the file extension. The new library is opened in the Browser.



Each icon in the Browser represents a single object. In this case the current icon represents the library you have just created.

To open an existing library

- Select the Library>>Open menu option on the Browser menu bar

or

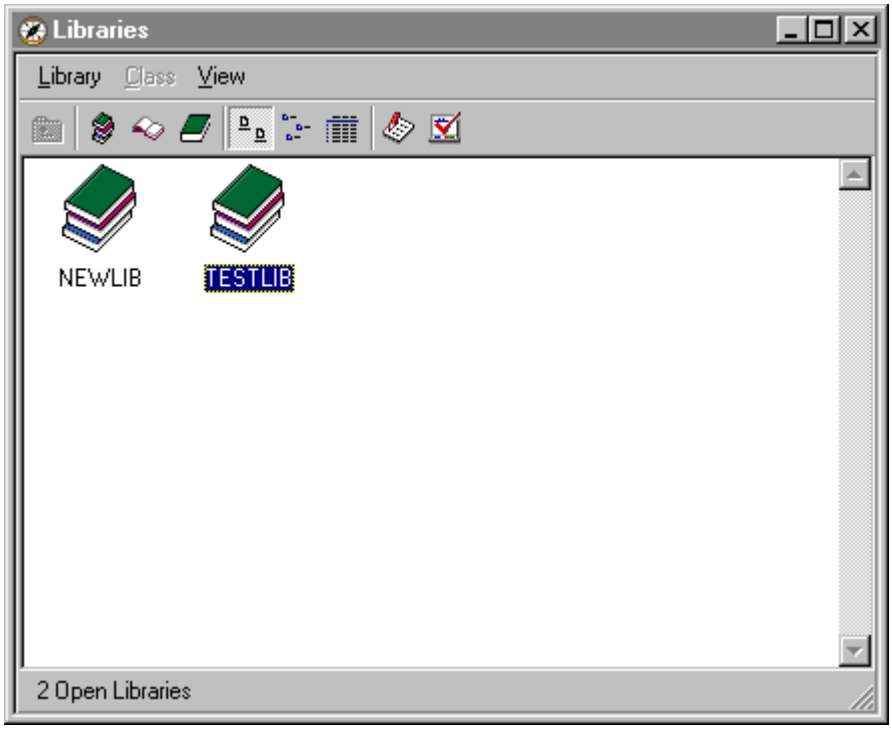
- Press Ctrl/Cmnd-O

or

- Drop its icon onto the Browser from anywhere in your system

The Library menu includes the names of any libraries you have opened recently. You can select a library name from this menu to open the library. You can remove an item in a Recent Items list by holding down the Shift key and selecting the item. In addition, if you select a file from a Recent Items list and the file cannot be opened, OMNIS puts up an OK message reporting the error, and indicating that the file will be removed from the list.

If you try to open a library created with OMNIS 7 Version 3, it will be converted. If you want to convert an older library please refer to the *OMNIS Studio Conversion* manual.



When you create or open more than one library, all the open libraries are shown in the Browser. You can view the classes in any library currently open in the Browser.

To close a library

- Select the library by clicking on its icon or name
- Select the Library>>Close menu option on the Browser menu bar

or you can

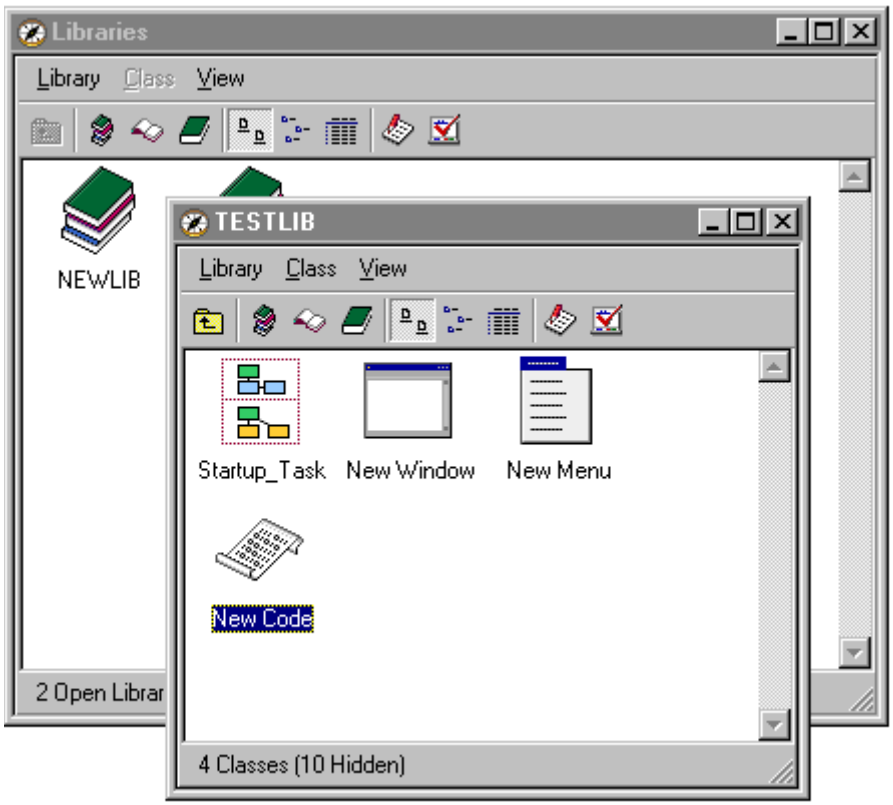
- Right-click on your library
- Select the Close option from the context menu

To view the classes in a library

- Double-click on the library icon

or

- Click on the library icon, and select the View>>Down One Level option on the Browser menu bar



When you first start OMNIS, the Browser is in multiple-window-mode which means that when you display the contents of a library a second Browser window is opened containing the classes in that library. In this mode if you double-click on another library a third Browser window is opened. Multiple Browser windows are useful if you want to copy classes from one library into another. To stop multiple Browser windows from opening you

can use the View>>Single Window Mode option on the Browser menu bar. In single-window-mode you can open one instance of the Browser only, so when you change levels the single Browser redraws rather than opening another window.

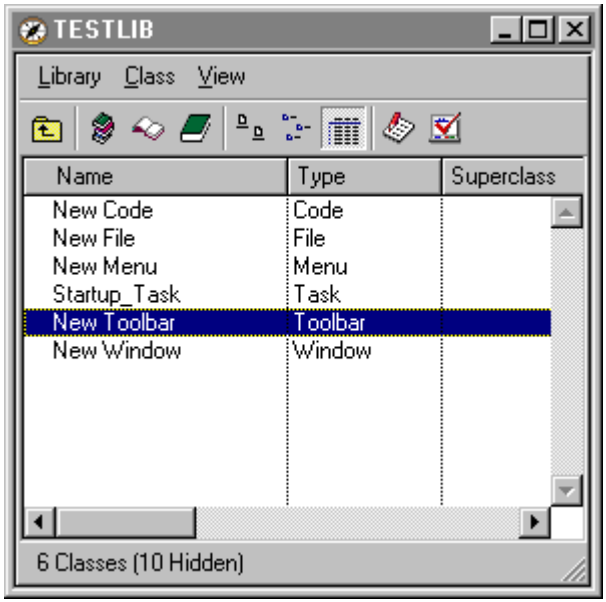
The default view for the Browser is to show Large Icons, but you can change its view using the View menu on the Browser menu bar or its context menu. You can select Small Icons or a Details view.

To change the Browser to Details view

- Select the View>>Details menu option on the Browser menu bar

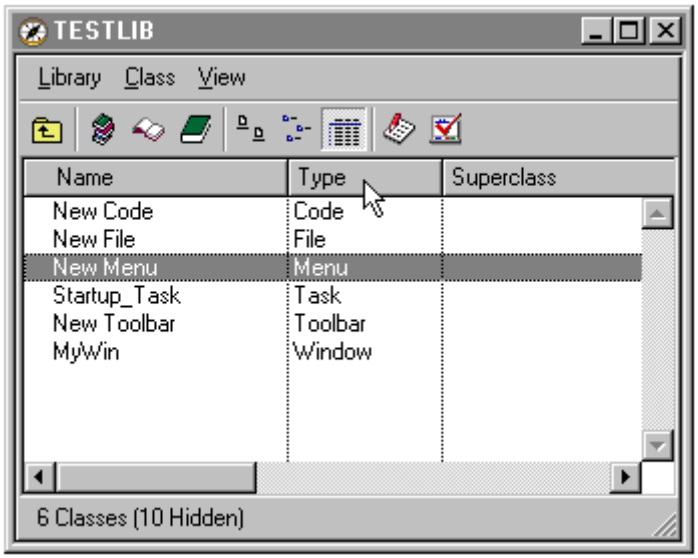
or you can

- Right-click on the background of the Browser, away from your library icon or name
- Select the Details option from the context menu



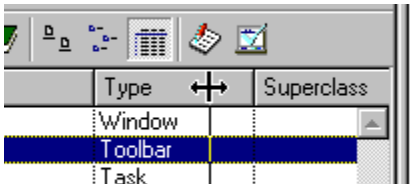
The Details view lists classes by Name, Type, Superclass, Description, Creation date, Modified date, and Size. The Details view for libraries shows the Name and Path of each open library.

You can sort the list of classes by category by clicking on the appropriate column header. For example, to sort the list by class type you can click on the *Type* column header. If you click repeatedly on the same header the sort order of the column switches between *ascending* and *descending* order.



Alternatively you can change the sort order of the list using the View>>Arrange by option on the Browser menu bar.

You can change the width of the columns in the Details view of the Browser by changing the width of the column headers: this applies to any of the tools that have a Details view. You can Shift-drag a column header to resize columns to the right; normally columns to the left are resized.

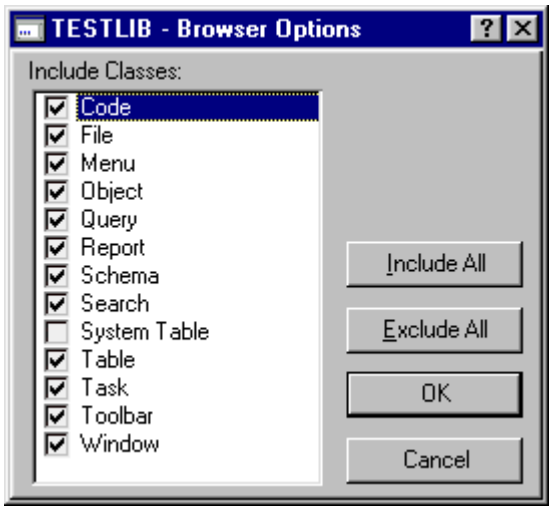


Hiding or Showing Classes in the Browser

You can hide and show the different types of classes using the Browser Options dialog.

To change the Browser options

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



You can 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.

The status bar on the bottom of the Browser tells you how many classes are displayed and how many are hidden in the current library.

Alternatively, while the Browser is the top window you can press Shift-Ctrl/Cmnd plus a letter key to show classes of one particular type. The following keys are available, including Shift-Ctrl/Cmnd-A to show all classes.

Shift-Ctrl/Cmnd	Displays...
A	all classes
W	windows only
F	files only
R	reports only
L	searches only
M	menus only
O	object classes only
J	tasks only
C	code classes only
T	toolbars only
S	schemas only
Q	query classes only
B	tables only
Y	system tables only

If a particular type of class is not currently visible in the Browser, OMNIS does not let you create that type of class. For example, if report classes are not shown in the Browser, you cannot create report classes, either using the Class>>New>>Report option on the Browser menu bar or by dragging a report class from the Component Store. In this case the Report option in the Class>>New menu is grayed out, and report classes will not drop onto the Browser.

The Browser options are stored for each library. Therefore when you open a library the options for that library will come into force, and only classes of one type may be shown.

Saving the Browser Setup

If you have changed the way the Browser is displayed and you want to save these changes for future use you can do this via the View menu on the Browser menu bar.

To save your Browser setup

- Select the View>>Save Window Setup menu option on the Browser menu bar

When you save the window setup all the settings in the View menu will be saved including the position of the Browser. When you re-open the Browser it will use these settings.

Component Store

The Component Store is an all-purpose repository for the objects you need to build your application. It contains class templates and wizards, field and background objects, and external components.

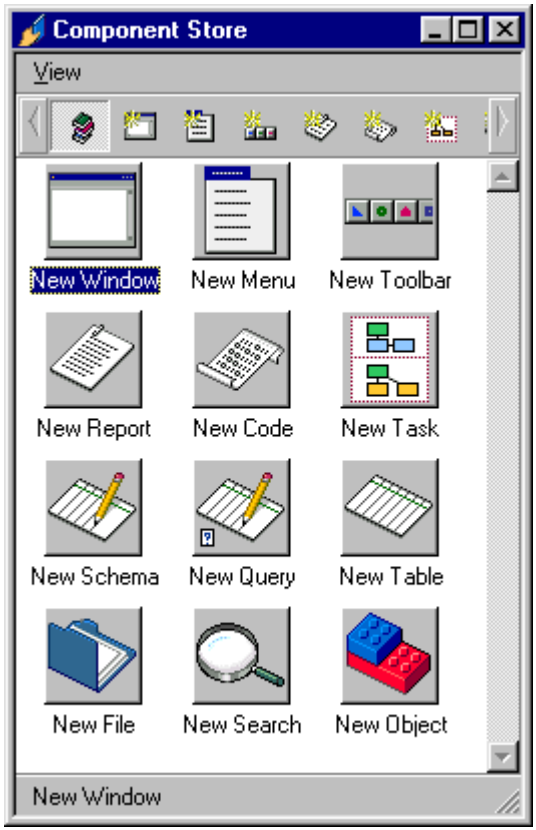
When you open a library, the Component Store opens automatically. You can create a new object in your library by dragging the object from the Component Store and dropping it onto the destination library in the Browser or the appropriate class design window.

To open the Component Store

- Select the View>>Component Store option from the main menu bar

or

- Press F3/Cmnd-3



Usually you won't need to open the Component Store since it will pop up when you open a library or any class design screen which uses it, including windows, reports, and toolbars.

When the Browser is the top window the Component Store contains template classes and wizards, but when you open a class design screen or editor the Component Store will show the objects or components available for that class type. For example, you open a window in design mode the Component Store displays window fields and components, or when you open a toolbar the Component Store shows tools and controls for toolbar classes.

The Component Store toolbar lets you select the category of objects shown in the main window. The tools available on the toolbar also change depending on context. For example, when you are creating classes in your library the toolbar shows tools for the different class types, and when you are designing a window, the toolbar displays tools for the different field and background objects.

You can change the view of the Component Store using the View menu. When you first open the Component Store it is shown in Large icons view, but using the View menu you can change it to Small Icons and you can hide or show the Text. You can experiment with large and small icons, sizing the Component Store window, and hiding and showing text, to achieve a palette-like Component Store, such as this

At any time you can save the setup of the Component Store using the View>>Save Window Setup option on the Component Store menu bar.

You can find general information about creating classes in the *Libraries and Classes* chapter. Individual classes are described in subsequent chapters in this manual.

Component Store Classes

When you first open a library the Default Classes button on the Component Store toolbar is selected showing a single template for each of the standard classes in OMNIS shown as a separate icon. The name of these classes defaults to "New <class>".

In addition to providing templates for the standard classes in OMNIS, the Component Store contains other templates and wizards for each type of class. Some templates are provided with OMNIS, but you can alter these or add your own to the Component Store. You can show the templates for each class by clicking on the appropriate button. You can move the mouse over each button to pop up a tooltip showing the class type. For example, the Window Classes button in the Component Store toolbar displays a set of window templates and wizards, and any other superclass templates currently loaded.

Changing the Default Classes

Under each type of class in the Component Store, the default class is shown on the template icon. The default class template or wizard is the one shown under the Default Classes button in the Component Store toolbar, and also the one used when you create a new class using the Class>>New menu option in the Browser. However, you can change the default template or wizard for a class.

To change the default template for a class

- In the Component Store toolbar, click on the type of class template you want to change, for example click on the Window Classes button
- Right-click on the class you want to make the default
- Select the Make Default option from the context menu

For example, under Window Classes you can make a window wizard the default class; therefore when you use Class>>New>>Window in the Browser the window wizard is called by default. When you return to the default display by clicking on the Default Classes button, the new default class is shown in the Component Store.

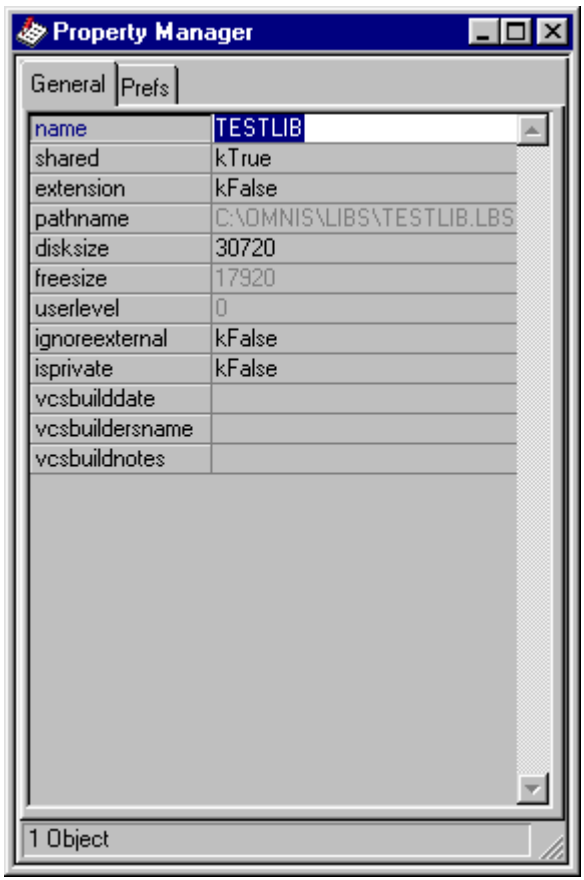
Adding Objects to the Component Store

You can add new objects to the Component Store by changing the Component Library. The View>>Show Component Library In Browser menu option in the Component Store lets you modify the Component Library. This is described in the *Library Tools* chapter.

Property Manager

The property manager lets you display or change the properties of the currently selected object. This could be a library, a window or OMNIS itself. The properties that are specific to each of the OMNIS classes are described in their individual chapters; OMNIS and library properties, or preferences, are described in the *Libraries and Classes* chapter.

The Property Manager will normally appear automatically when needed; if it is not visible you can display it either by selecting the View>>Property Manager menu item from the main menu bar, by pressing F6/Cmnd-6, or from an object's context menu. For example, you can click on your library and the Property Manager displays its properties.



Changing Object Properties

You can change the properties of any object in OMNIS using the Property Manager. You can Right-button click under Windows or Ctrl-click under MacOS on most objects in OMNIS to open a context menu containing a Properties option. When you select the Properties option, the Property Manager will open showing the properties of the current selected object. At other times the Property Manager will popup automatically.

The Property Manager context menu has a ‘Multi-Line Tabs’ option that lets you toggle between a scrollable, single line tab pane, or a multi-line tab pane. The default depends on the operating system you are using.

To change a property of an object

- Right-click on the object to open its context menu
- Select the Properties menu item
- Select the property you want to change
- Type in a new value

or, to change a boolean or True/False property

- Select the property you want to change in the Property Manager
- Double-click on the current value to toggle the value

or, to cycle through all possible values

- Select the property you want to change in the Property Manager
- Repeatedly double-click on the property value

or, to select a value with the mouse

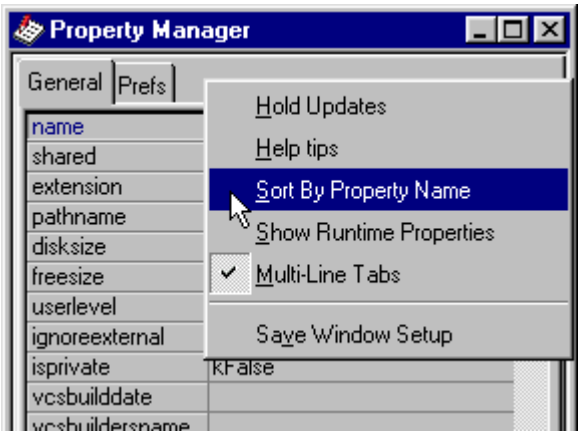
- Click on the down arrow in the property value field
- Select a value from the droplist

Sorting Properties in the Property Manager

The properties of an object are shown in the Property Manager in functional order by default, but you can list them alphabetically using the View>>Sort by Property Name option in the Property Manager context menu. This manual uses the functional order, but once you get to know object properties you may like to view them in the Property Manager in alphabetical order.

To sort properties by name

- Right-click on the Property Manager, away from a property name either at the top next to the tabs or at the bottom of the Property Manager window



- Select the Sort by Property Name menu option from the context menu

The other options in the Property Manager context menu affect the behavior of the Property Manager. If set the Hold Updates option stops the Property Manager updating its contents automatically when you click on another object. For example, you can click on a window class to show its properties in the Property Manager, select the Hold Updates option, then click on a field in the window, and the Property Manager still displays the properties of the window. Most of the time however you'll want this option turned off so the Property Manager updates itself automatically.

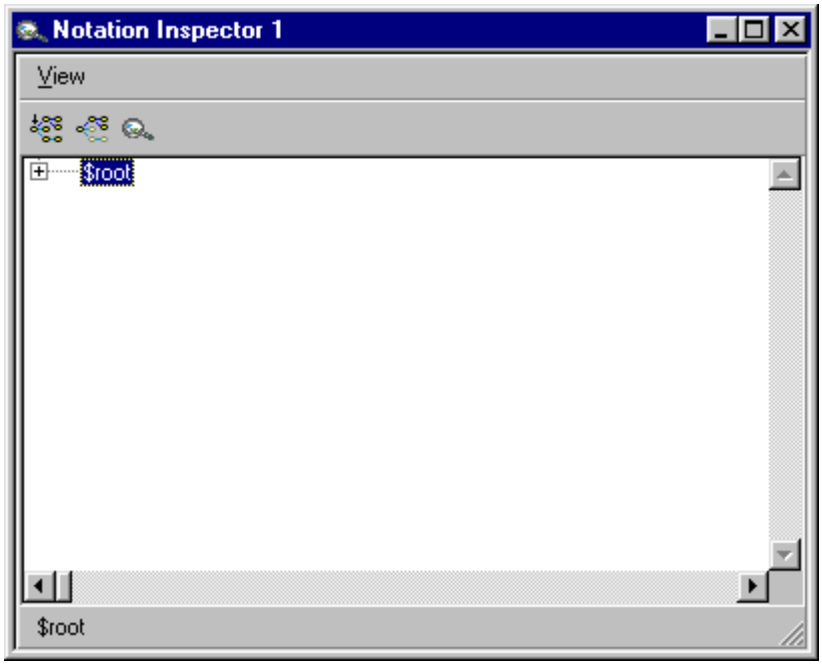
The Help tips option in the Property Manager context menu displays short descriptive help messages for each property in the Property Manager. The Show Runtime Properties option lets you view properties that are normally available in runtime only, that is, properties of an instance rather than a design class. When runtime properties are visible in the Property Manager the methods for the instance are also shown. You cannot set runtime properties or use methods shown in the Property Manager, they are there as a convenient reference when you are writing code.

Notation Inspector

The Notation Inspector lets you view the OMNIS object tree from the $root object down. It is particularly useful for viewing the contents of your library and finding the right notation for a particular object or group of objects in your library. For example, you can get the notation for an object on a design or open window using the Notation Search tool on the Notation Inspector toolbar. When you click on an object or group in the Notation Inspector the Property Manager will display its properties and methods.

To open the Notation Inspector

- Select the View>>Notation Inspector option on the main menu bar, or press F4/Cmnd-4



When the Notation Inspector opens it shows $root which contains all the objects in the OMNIS object tree including all your open libraries and their contents. It also includes all the objects and groups created at runtime when you run your application. You can expand each branch of the tree to show the contents of an object or group.

To expand or collapse a node in the Notation Inspector

- Click on the expand or collapse icon, the + or - icon

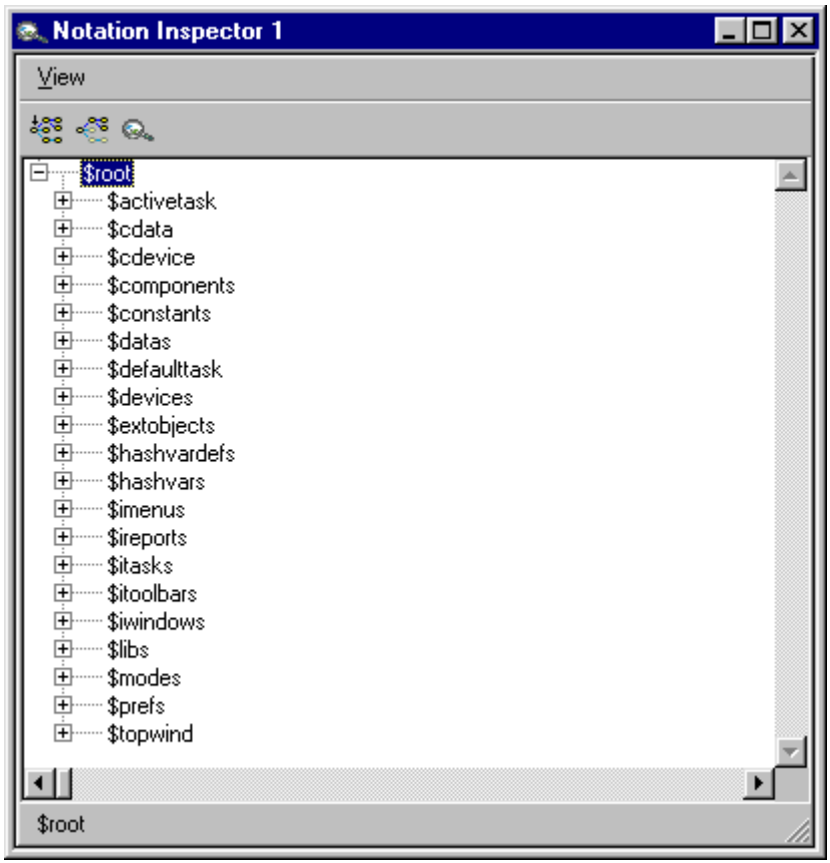
or you can

- Double-click on an object or group name

When you expand and collapse the tree OMNIS will scroll the Notation Inspector automatically. You can view all your open libraries in the $libs group, and you can view the contents of a particular library using the Notation Inspector.

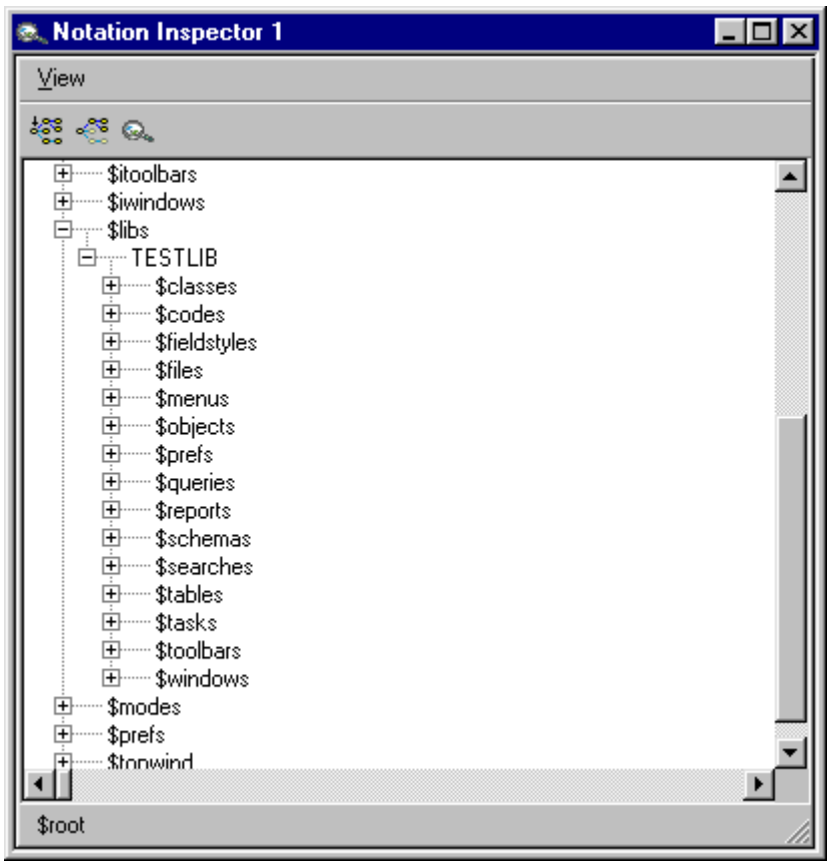
To view the contents of your library

- Click on the expand icon at the $root of the tree, to show the contents of OMNIS



- Click on the expand icon of the $libs group, to show all your open libraries

- Click on the expand icon for your library, to show the contents of your library



All the different types of class in your library are shown in Notation Inspector within their respective object group. For example, all the window classes in your library are contained in the $windows group. Likewise all the toolbar classes are contained in the $toolbars group. Note that the $classes group contains *all* the classes in your library. Also note that a group may be empty if your library does not contain any classes of that type.

To find a class in the Notation Inspector

- Show the contents of your library in the Notation Inspector, as above
- Open the group containing the class you want; for example to find a particular window class, open the $windows group and click on the window name

The status bar at the bottom of the Notation Inspector shows the notation for the currently selected object or group of objects.

Having found the object you're interested in, you can copy its full notation to a calculation or you can place its notation on the clipboard ready to paste anywhere in your library. For example, you can drag the item into a calculation field in the method editor.

To copy the notation for an object

- In the Notation Inspector find the object you're interested in, as above
- Click on the object and press Ctrl/Cmnd-C
- Go to an entry field or Calculation field and press Ctrl/Cmnd-P to paste the complete notation string

or

- Locate the object you're interested in
- Drag the object from the Notation Inspector into a calculation field

or to copy the object or group name only

- Hold down the Shift key and drag the object into your calculation field

When you Shift-drag from the Notation Inspector, the object or group name is prefixed with a dot which is suitable for appending to a notation string you may have already entered into a calculation field. Note that when the Property Manager displays properties with their $ names, it also lets you drag-copy either complete notation strings or Shift-drag-copy single property names into calculation fields.

Notation Inspector Toolbar

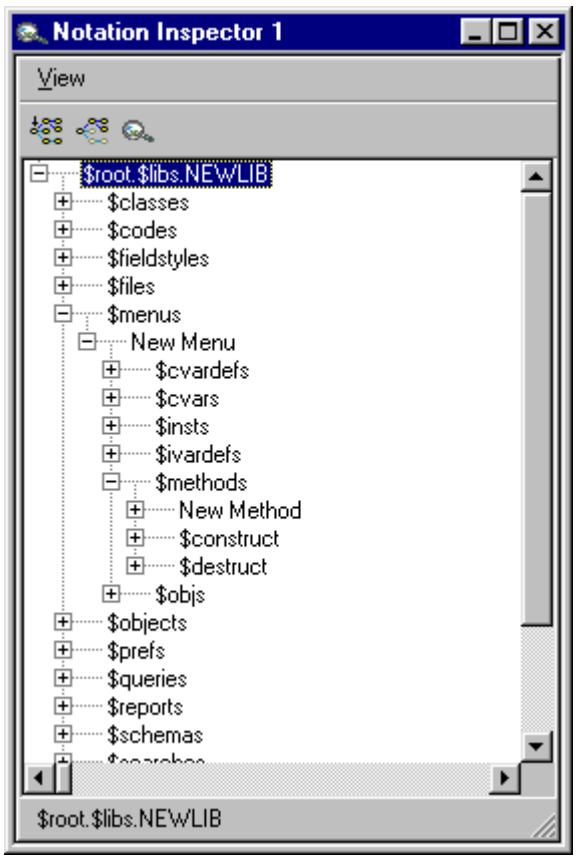
You can install the Notation Inspector toolbar from its View menu. This toolbar lets you view the full notation tree, or make the current object the root or base of the tree. Also the toolbar contains a button to get the notation for a window, report, or toolbar object. You can pass the mouse over each button to popup a tooltip if you're not sure what a button does.

When you have located a particular library in the Notation Inspector you may want to hide the other objects in OMNIS and make that library the root of the notation tree.

To make a library the root of the notation tree

- Click on $root, to show the contents of OMNIS
- Click on $libs, to show all the open libraries
- Select your library to highlight its name

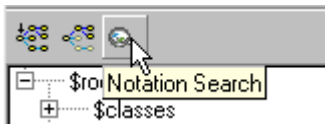
- Click on the Make root button on the Notation Inspector toolbar



The Notation Inspector will refresh itself with your library at the root of the tree.

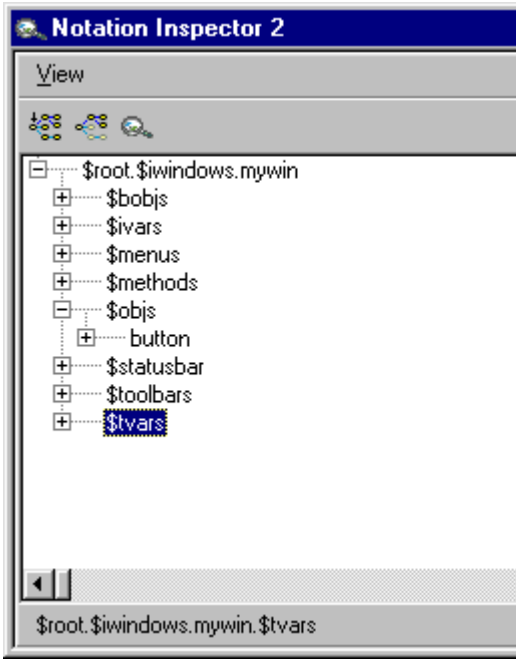
To find the notation for a particular object

- Click on the Notation Search button on the Notation Inspector toolbar

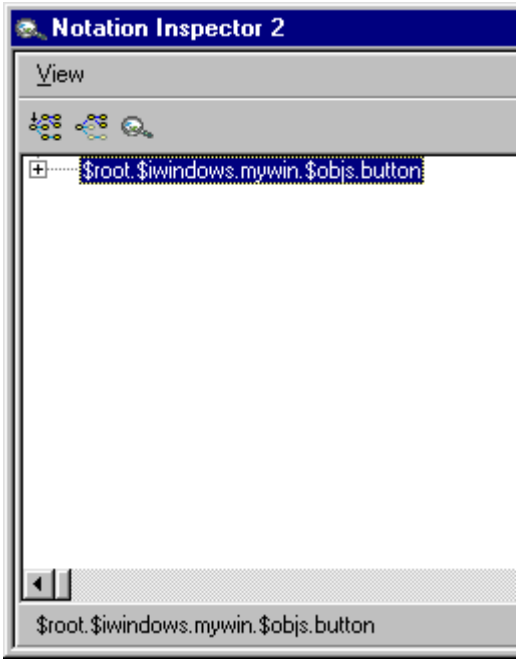


- Click on the object in your window, report, or toolbar object

You can get the notation for a window object in design or runtime mode, also report objects and toolbar objects. The Notation Inspector will refresh itself showing the notation for the object you clicked on. The object becomes the root of the tree, so you can expand the tree to view its contents.



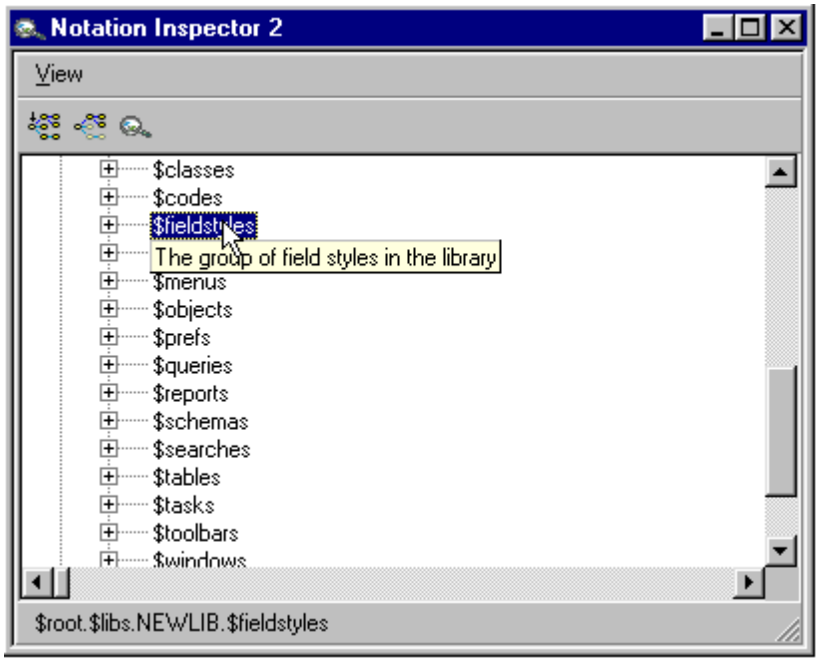
Notation Inspector showing the notation for a window instance and its contents



Notation Inspector showing the notation for a field on a window

Once you've identified the notation for an object in the Notation Inspector, you can copy its full notation as described above. You can show the whole of the notation tree again by clicking on the Show Full Notation Tree button on the Notation Inspector toolbar.

You can show helptips for each item in the Notation Inspector. In this context, each helptip provides the description for each object or group in the Notation Inspector. You can enable helptips from the Notation Inspector View menu or its context menu.



Notation Inspector showing the helptip for the $fieldstyles group

Viewing Properties and Methods

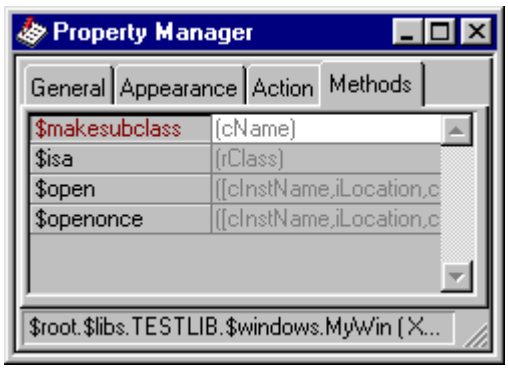
You can use the Notation Inspector in conjunction with the Property Manager to view the properties and methods for an object or group in the object tree.

To view object properties from the Notation Inspector

- Open the Notation Inspector and expand the tree to find the object or group you're interested in
- Click on the object or group to popup the Property Manager; if the Property Manager is already open you may need to press F6/Cmnd-6 to bring it to the top

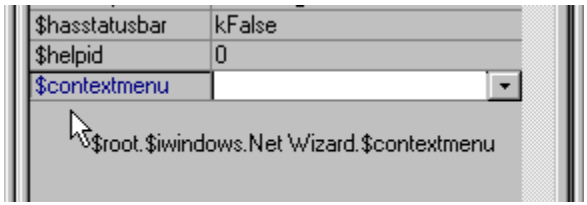
Note that the properties shown in the Property Manager include the dollar sign; you will see these only when you view object properties via the Notation Inspector, otherwise properties are shown with their simple names without the dollar sign.

For object groups you will see their standard method names and parameters, but for other objects you may need to click on the Methods tab to display their methods. For example, a window class has the following standard methods



The Notation Inspector displays some properties in the Property Manager as “(Context sensitive)”. This means that the property is supported for the item, but its value cannot be displayed in the Property Manager, since it requires a context, for example an executing method, in which to be evaluated.

Dragging Properties, Methods, and Notation



As already described, you can drag the full notation for an item from the Notation Inspector to anywhere in the method editor. In addition, you can drag properties and methods from the Property Manager to any calculation field in the method editor, provided the Property Manager is populated via clicking on an item in the Notation Inspector (in this case, property names include a dollar sign). In both cases, the full notation string for the item is dragged (as shown), unless you hold down the Shift key, in which case the object or property name prefixed with a dot is dragged.

Inheritance Tree

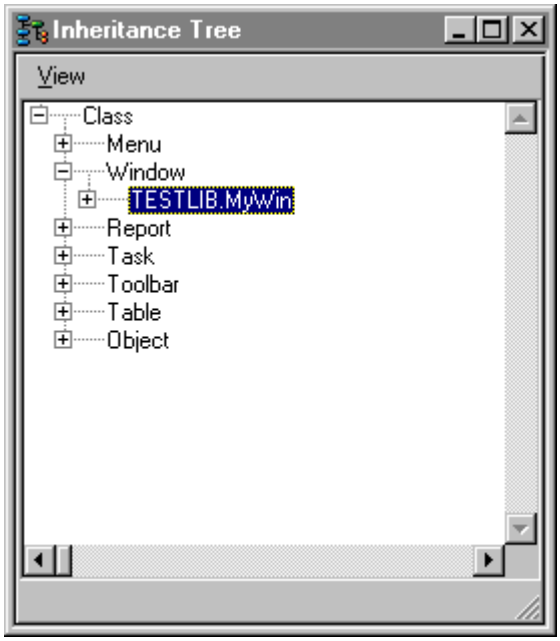
The *Inheritance Tree* shows the superclass/subclass structure of the classes in your libraries.

To view the Inheritance Tree for a class

- Click on a class in the Browser
- Select View>>Inheritance Tree from the main menu bar or press F5/Cmnd-5

or

- Right-click on the class and select Inheritance Tree from its context menu



See the *Object Oriented Programming* chapter in the *OMNIS Programming* manual for information on using inheritance in OMNIS.

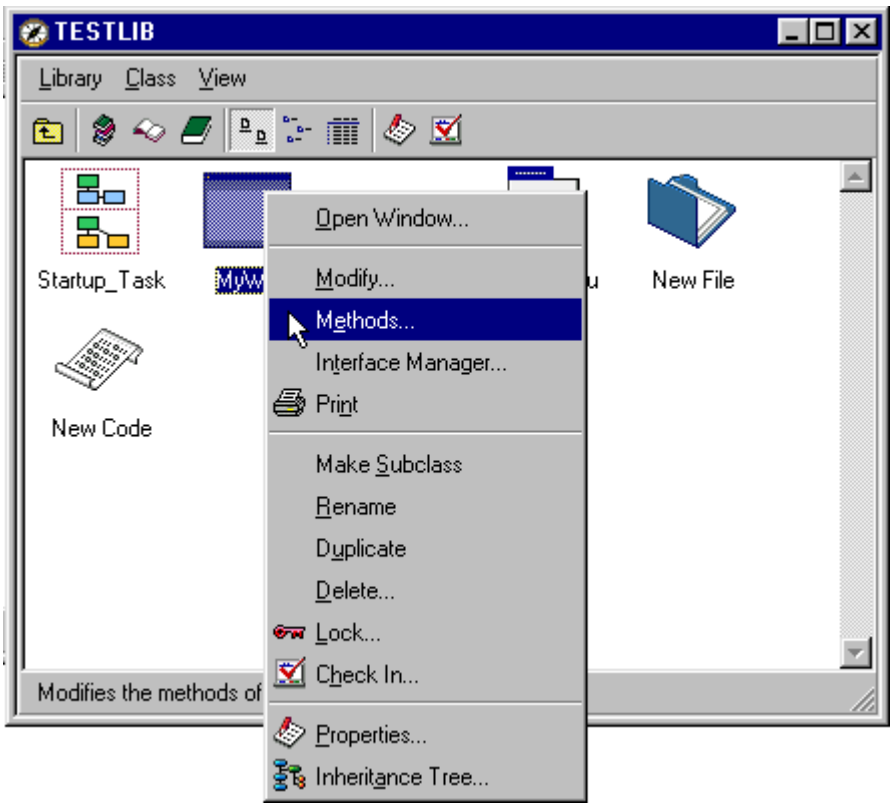
Method Editor

You add variables and methods to the objects in your library using the *method editor*. You can also debug the methods in your library using the OMNIS debugger, which is an integral part of the method editor. You can access the method editor in a number of ways, depending on the type of object you're working on and where you are in OMNIS.

For further details about adding variables and methods to the objects in your library, see the *Variables and Methods* chapter. All aspects of programming OMNIS methods are covered in the *OMNIS Programming* manual.

To open the method editor for a class

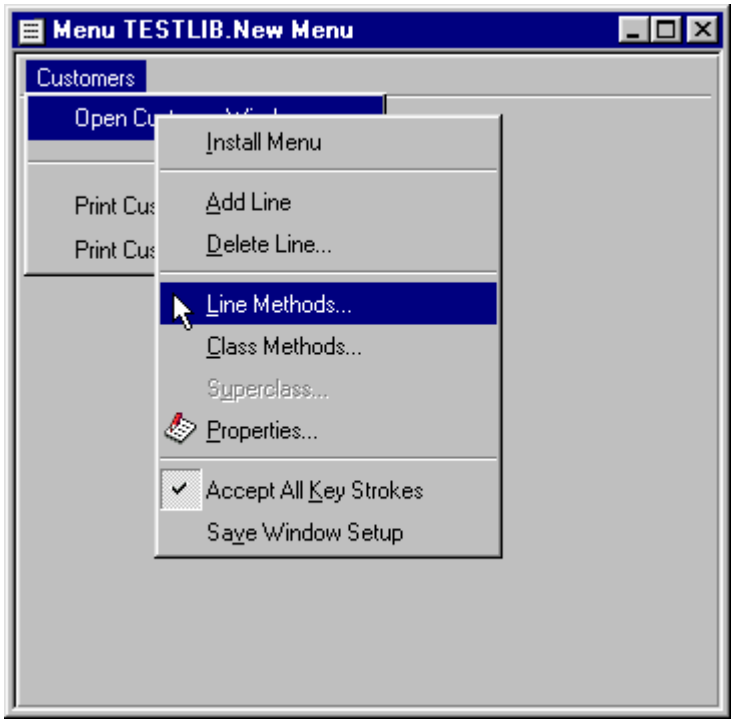
- Open your library and view its classes in the Browser
- Right-click on the class
- Select the Methods option from the context menu



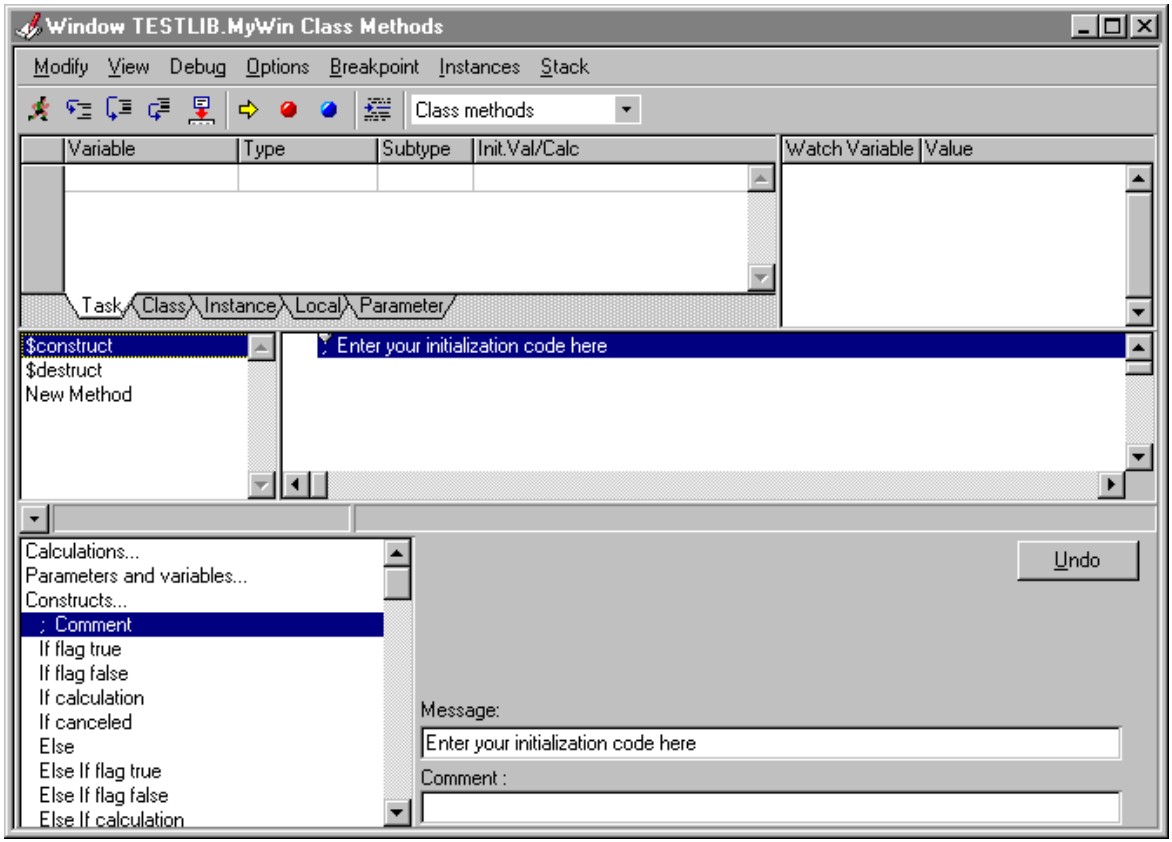
or for window, report, menu, and toolbar classes you can

- Open the design screen for the class from the Browser
- Right-click on the background of the class design screen
- Select the Methods option from the context menu

For example you can Right-click on the background of a window class or on one of its fields, and select either the Field or Class Methods option from the context menu. For window and report classes, you can add methods to a single field or to the class itself. For toolbar classes you can add methods to a Tool and the Class, and for menu classes you can add methods to a particular menu Line and the Class. Alternatively, when editing a window, menu, toolbar, or report class, you can press F8/Cmnd-8 to open the *class methods*, and Shift-F8/Cmnd-8 to open the *field methods* assuming a field or data object is selected.

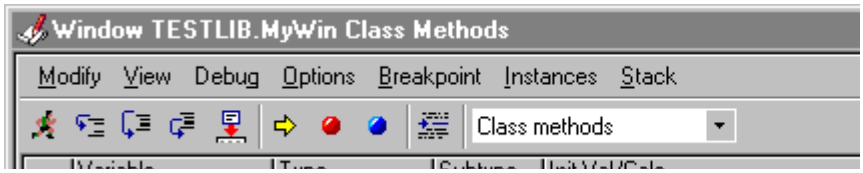


Code, task, table, and object classes contain methods only, so when you modify these classes you go straight into the method editor. To add methods to these classes you can double-click on the class name in the Browser.

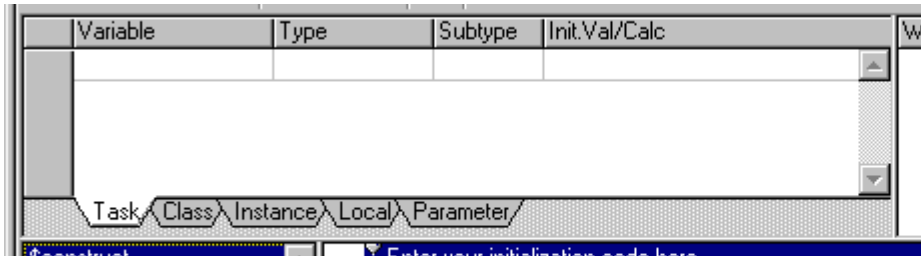


The method editor has several different areas, each doing a different job. These are

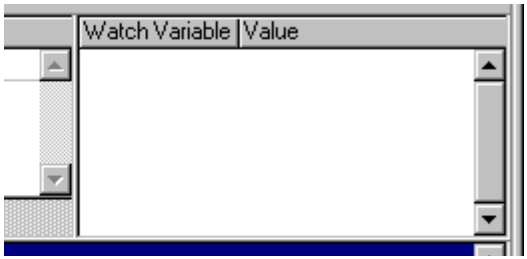
– **Menubar** and **Toolbar**
let you add, edit, and execute methods, in addition to debugging the methods in your application



– **Variables** panes
lets you add variables to the class or method

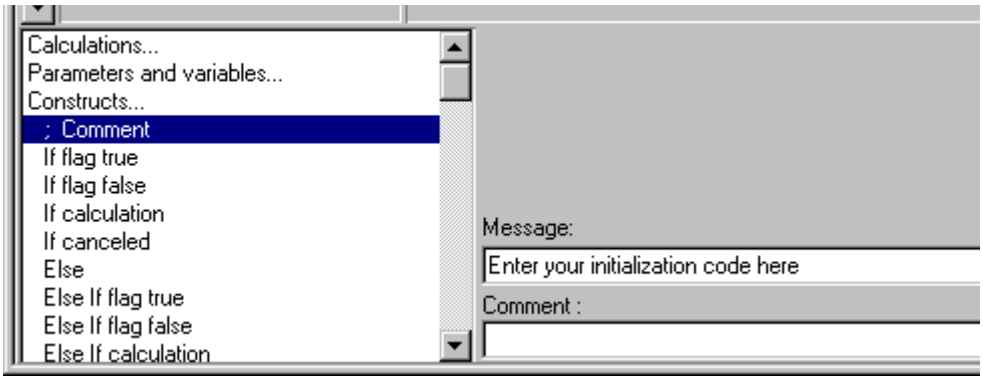


– **Watch variables** pane
lets you monitor variable values while debugging



– **Method names** and **Method definition**
lets you add methods to the object, lists existing methods for the object, lets you add the code for a method, or view the code for an existing method

– **Command palette**
lists all the commands, and the parameters for the current command



Interface Manager

The Interface Manager displays the public methods and properties for window, menu, toolbar, report, task, table, and object classes. You can view the Interface Manager from several places in OMNIS, including the Browser, method editor, and from various context menus.

To view the Interface Manager

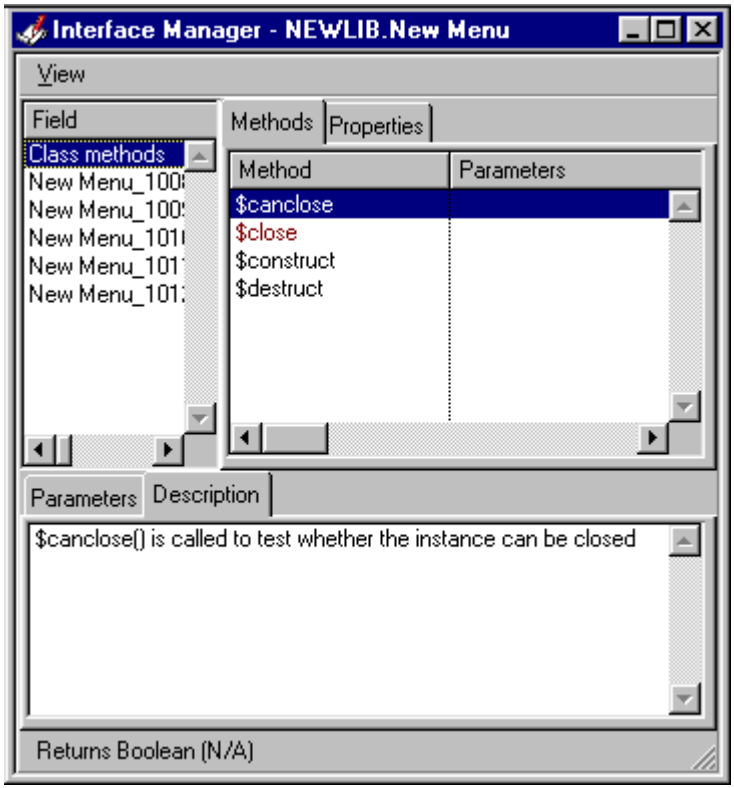
- Click on the class in the Browser
- Select Class>>Interface Manager from the Browser menubar

or you can

- Right-click on the class in the Browser and select Interface Manager from the context menu

or from the method editor

- Open the method editor for the class
- Select View>>Interface Manager from the method editor menubar



For more details about using the Interface Manager, please see the *Object Oriented Programming* chapter in the *OMNIS Programming* manual.

Catalog

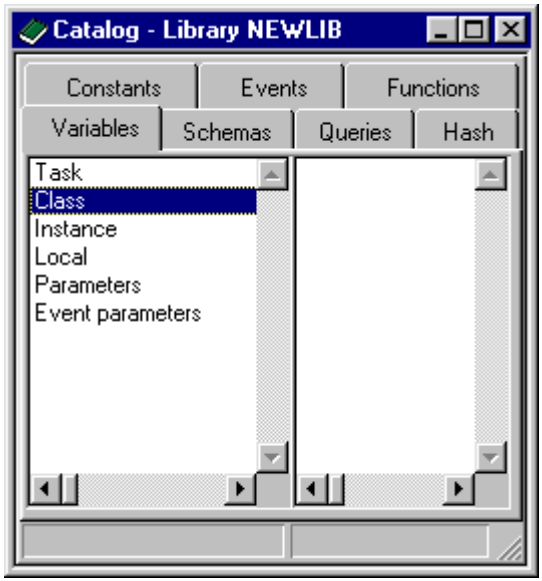
The Catalog lists all the variables, schema and query class columns, and file class fields in your library, in addition to listing all OMNIS functions, constants, event codes, and hash variables. When you have found an item in the Catalog, you can enter its name into a calculation field by double-clicking on it or dragging it out of the Catalog.

To open the Catalog

- Select the View>>Catalog option on the main menu bar

or

- You can press F9 under Windows, or Cmnd-9 under MacOS anywhere and at any time in OMNIS to open the Catalog

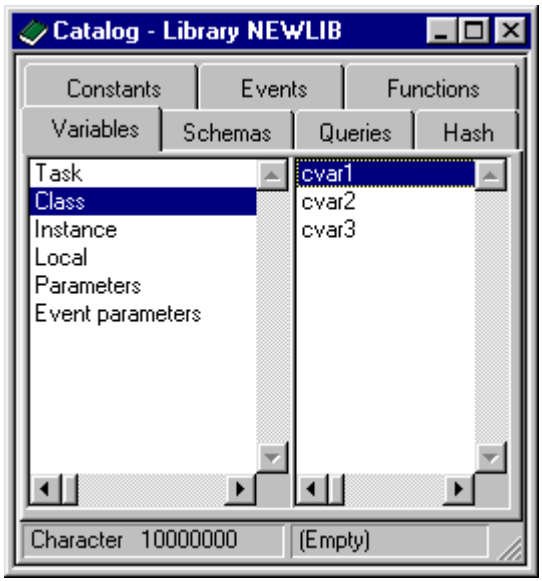


The Catalog lists all the variables for the current object including task, class, instance, local, and parameter variables, and also event parameters. For example, if you are working in a code class you can click on Class under the Variables tab and the Catalog will display all the class variables for that code class. Alternatively, if you are working in a window class the Catalog will show the class and instance variables for that window class. When you select a particular method in the method editor, the Catalog will list the local and parameter variables for that method.

The Catalog also shows all the file classes in the current library. When you click on a file class name, in the left column under the Variables tab, the Catalog will display all the fields for that file class, in the right column.

To select a variable from the Catalog

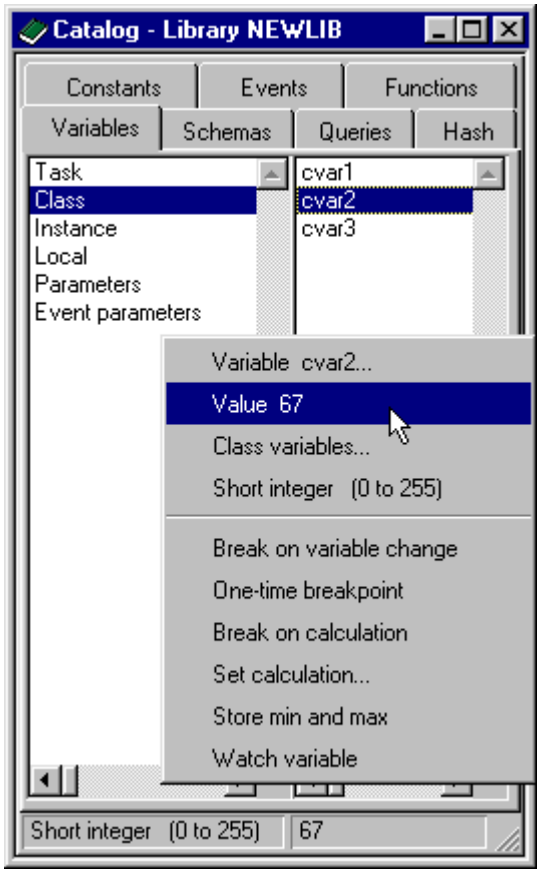
- Open a method editor for the class and go to where you want to enter a class variable name, in a *Calculate* command for example
- Open the Catalog by pressing F9/Cmnd-9
- Click on the Variables tab
- Select Class in the left column of the Variables pane



- Either Double-click on the name of the class variable to enter it into the edit field, or drag it from the Catalog and drop it on the field; note that you *cannot* use Copy and Paste from the Catalog

The name of the variable will be entered at the current position of the cursor. You can do this wherever you need to enter the name of a variable or field. If you are entering a comma-separated list of parameters in a method or command, you can hold down the Shift key to append a comma to the item.

If you want to know the current *value* of a variable or field you can Right-click on its name in the Catalog. The Variable popup menu will show the variable name, its value, and type.



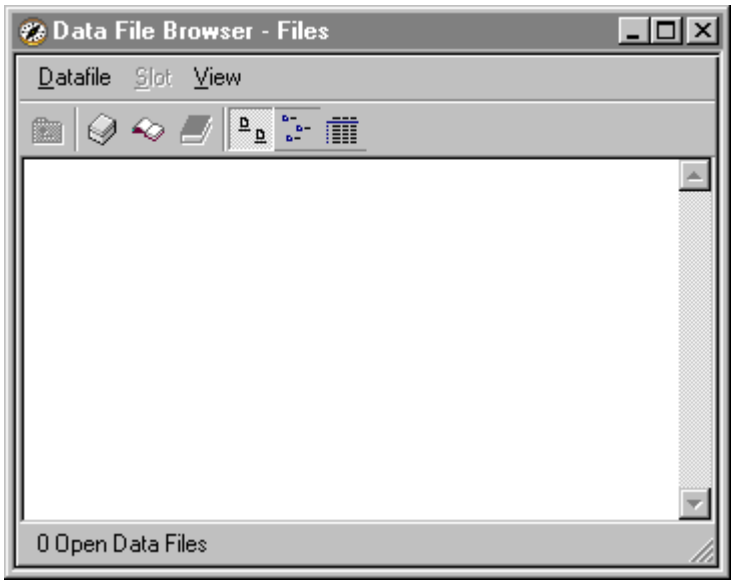
You can popup this menu wherever the variable name appears in OMNIS, you don't have to be in the Catalog. The first option in this menu opens the Variable value window in which you can view and change the current value of the variable or field, unless the variable is a Binary or an Item reference, in the latter case the option opens the Notation Inspector.

Data File Browser

You store local data in an OMNIS *data file*. 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 data, and create, rename, or delete data file slots.

To open the Data File Browser

- Select the View>>Data File Browser option on the main menu bar

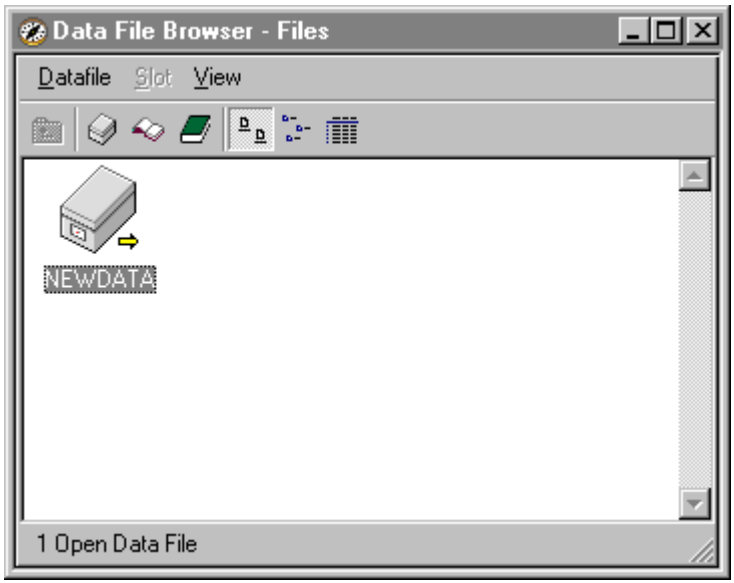


The Datafile menu lets you create new data files, open an existing one, or close the currently selected data file. Also it contains options for checking your data; for details about these features see the *OMNIS Data File*s chapter in the OMNIS Programming manual.

The View menu lets you change how the Data File Browser is displayed and lets you save the current window setup. It also lets you hide or show the menu bar, toolbar, and status bar. The status bar displays short help messages for the options in the Data File Browser. Like the standard Browser you can Right-click on the Data File Browser to open the View context menu.

To create a new data file

- Select the Datafile>>New option from the Data File Browser menu bar
- Enter a name for the new data file, including the file extension .DF1, and click OK



To open an existing data file

- Select the Datafile>>Open option from the Data File Browser menu bar
- Locate the data file in the Open Data File dialog and click OK

or

- Drop the data file's icon on to the Data File Browser from anywhere in your system

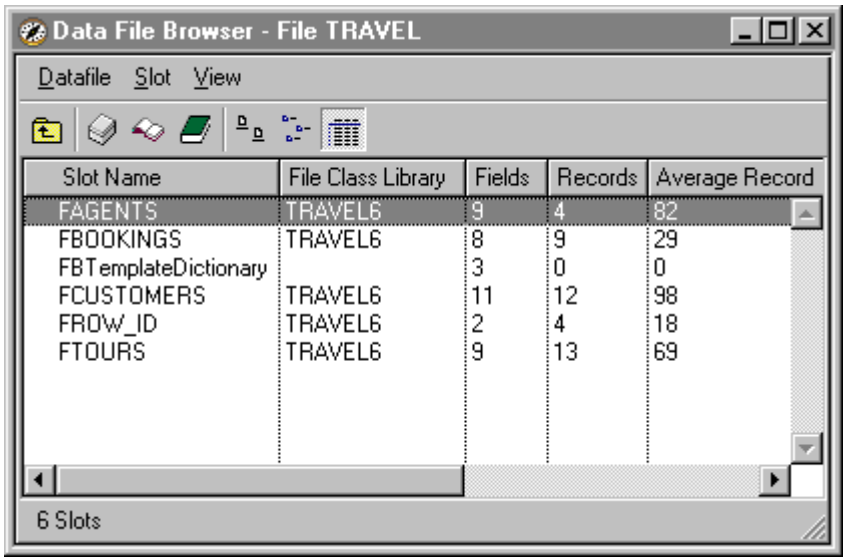
or

- If the data file has been opened recently, you can select it off the list of recently opened datafiles on the Datafile menu

You use the View menu in the Data File Browser to view the slots in an open data file.

To view the data slots in an open data file

- Switch the Data File Browser to Details view, and click on the data file you want to examine
- Select the View>>Down One Level option in the Data File Browser
- Click on the Data Slots, and select the View>>Down One Level option again



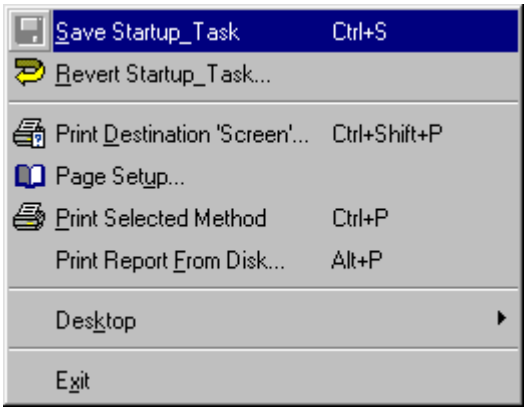
Generally there should be one slot for every file class in your library. The Data File Browser shows you the number of fields and records (or rows) for each slot. You can print a listing of all data slots in an open data file by selecting File>>Print. The Slot menu lets you manipulate individual slots.

Reorganizing, Checking, and Updating data files is discussed in detail in the *OMNIS Data Files* chapter in the *OMNIS Programming* manual.

Standard Menus and Toolbars

This section describes the standard File, Edit, View, Tools, Window, and Help menus. It also describes the IDE toolbars and desktop mode.

File menu



The File menu lets you save the current class design window, and set the current print destination. It also lets you change the desktop mode.

The **Save** <Class> option (Ctrl/Cmnd-S) saves the class you are currently working on. If a class design window is not currently selected this option is grayed out. The **Revert** <Class> option rolls back any changes you have made to the class you are currently working on.

The **Print Destination** (Shift-Ctrl/Cmnd-P) dialog lets you set the destination of the current output. Reports are sent to the Screen by default, but you can choose another destination from this dialog. For further information, see the *Report Classes* chapter.

The **Page Setup** option opens the standard Print Setup dialog for the current OS.

The **Print** option (Ctrl/Cmnd-P) prints the current object if appropriate: for example, a report class, or when you are working in the method editor, it prints the currently selected method or methods.

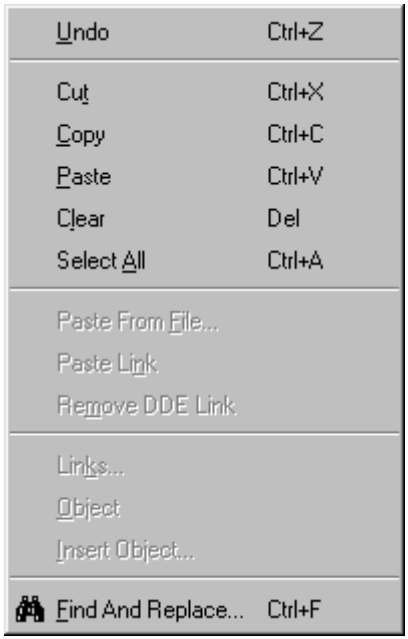
The **Print Report from Disk** (Alt-P/Option-Cmnd-P) option lets you select and print a report previously stored on disk.

The **Desktop** submenu lets you select the current mode for the IDE from the three modes *Design*, *Runtime* and *Combined*. In Design mode the standard menus, such as File, Edit, View, and Tools, plus the IDE toolbars are visible. In Runtime mode all these are hidden and only user menus, data entry windows and toolbars defined in your library are visible. Also in runtime mode there is a cut-down version of the File and Edit

menus on the main menu bar. Being able to switch to runtime mode lets you see exactly how your application will look when the user runs it. In Combined mode (the default) all design and user menus, windows, dialogs, and toolbars are visible. When you select the Runtime option, the Desktop toolbar is installed so you can get back to the Combined mode.

The **Exit/Quit** option closes OMNIS and any open libraries saving all classes.

Edit menu



The **Undo** option (Ctrl/Cmnd-Z) reverses the last cut, paste, move, or text/graphic edit. For example, if you have just moved an object the undo option will read Undo Move.

The **Cut** option (Ctrl/Cmnd-X) removes the selected object(s) on the window or report, or item of data, to the clipboard. The OMNIS clipboard format is not compatible with other applications, but text and graphics can be cut and pasted during window or report design and during data entry.

The **Copy** option (Ctrl/Cmnd-C) copies the selected object(s) or text to the clipboard; the source remains unchanged.

The **Paste** option (Ctrl/Cmnd-V) takes information stored on the clipboard, whether an object or an item of data, and places it where you are currently working. This option is grayed out when the clipboard is empty or when the data on the clipboard is not compatible with where you are working.

The **Clear** option (Del) removes the selected screen object(s) or item of data; the information is not stored on the clipboard.

The **Select All** option (Ctrl/Cmnd-A) highlights the whole of the current window, field or text edit area; the selected area can then be cut or copied in the usual way.

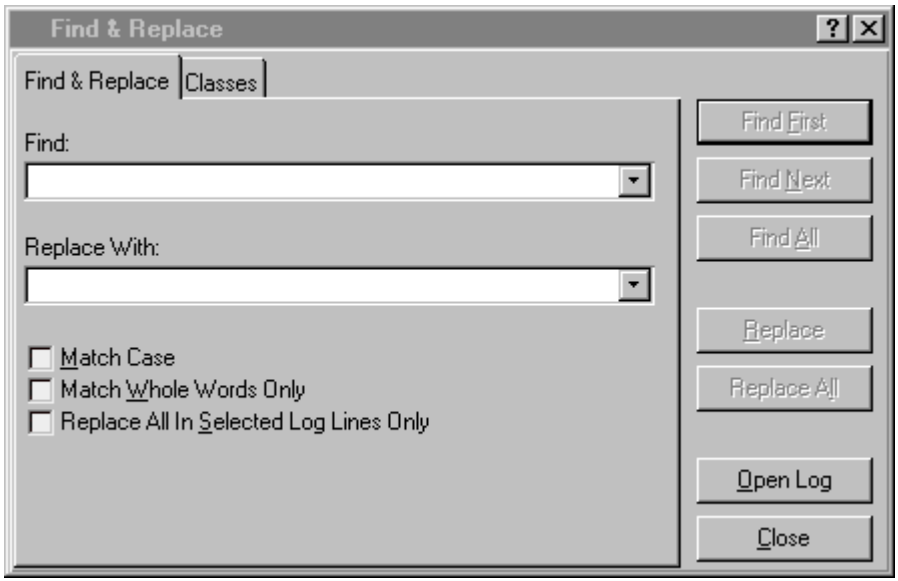
The **Paste From File** option lets you read in text or pictures in Windows bitmap (.BMP), metafile (.WMF) or MacOS PICT format. You can paste text or pictures into fields on windows and reports, or the background of a window or report.

DDE and OLE (under Windows only)

The **Paste Link** option places a DDE link from Excel or Word, for example, in the current field. The **Remove DDE Link** option deletes the link in the current field. The **Links** option applies to OLE only, and displays a list of current links from which you can choose a link. The **Object** option shows the name of the current linked or embedded object. The **Insert Object** option inserts an embedded object into the current field.

Find and Replace

The Find & Replace tool lets you search through a class or library, or a number of classes or libraries, to find a particular text string. You can selectively replace occurrences of an item or replace all items. The **Find And Replace** option under the Edit menu (or Ctrl/Cmnd-F) opens the Find & Replace dialog.



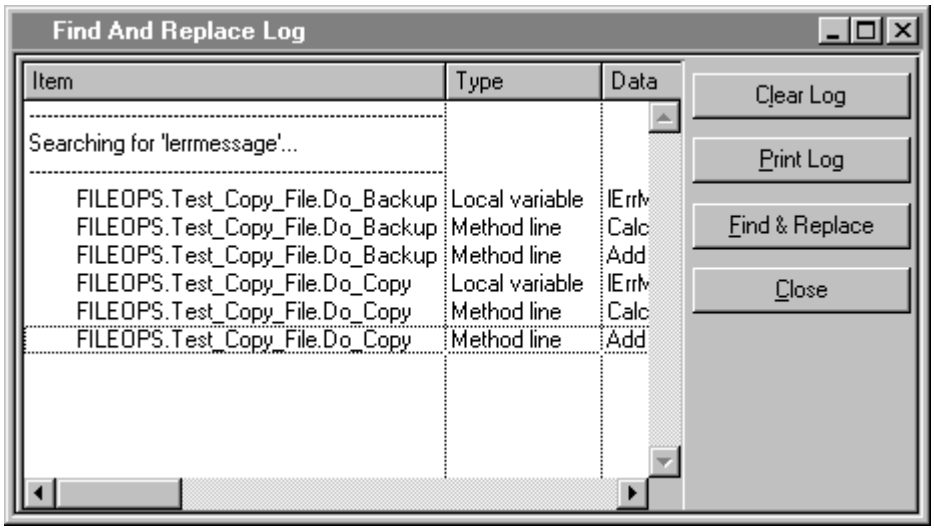
You enter the item you want to find in the Find entry box, and check the Match Case and/or Match Whole Words Only boxes as appropriate.

On the Classes tab you can select the libraries and classes in which you want to perform the find and replace. For a single library you can select some or all of the classes in the library. File classes appear at the top of the list and are searched first. If you select more than one library under the Classes tab, all classes in all selected libraries are searched. You can also choose to search just all the classes in a particular class type.

If you click on the Find First button OMNIS will jump to the first occurrence of the text string in your selected classes or libraries. For example, if the specified item is found in a method OMNIS will open the class containing the method with the found item highlighted. Find Next will jump to the next occurrence of the text string, and so on. The Find All button opens the Find log and finds all occurrences of the specified item in all the classes or libraries you selected.

If you click on the Replace button OMNIS will find and replace the first occurrence of the specified item, or Replace All will find and replace all occurrences of the item. If you have previously performed a find all, you can select particular lines in the log, check the Replace All In Selected Log Lines Only check box, and click on the Replace button to replace the selected references only.

You can interrupt a find and replace operation at any time by pressing Ctrl-Break under Windows, or Cmnd-. under MacOS. When you perform a find and replace all, OMNIS opens the Find And Replace Log.

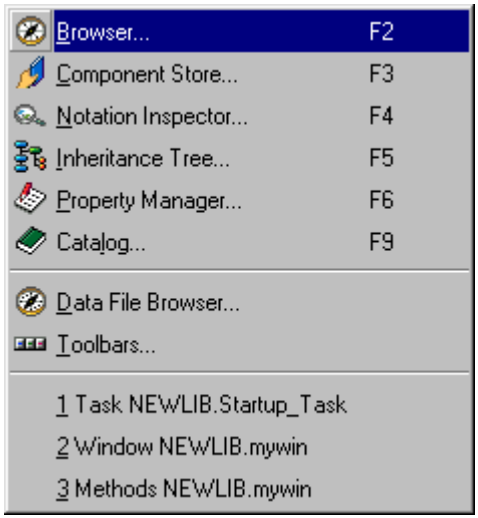


The first column in the log lists the class containing the found item. The second column tells you the Type of object, such as a method or property, and the third column contains the actual data. You can double-click on a line in the log to jump to the data.

When you rename certain objects in your library, OMNIS will replace all references to the object automatically. For example, if you rename a class variable in the method editor

OMNIS will replace all references to the variable within the class automatically. However if you try to rename most other types of object, such as renaming a class in the Browser, OMNIS will prompt you to find and replace references to the object. If you answer Yes to the prompt, OMNIS will open the Find & Replace tool and the Find log which lets you monitor the find and replace, or control whether or not certain references are replaced.

View menu



The **View** menu on the main OMNIS menu bar lets you access all the main development tools such as the Browser and Component Store. For a description of individual tools, see earlier in this chapter. Many of the OMNIS tools also have View menus that let you change the view or behavior of the tools.

The **Browser** option (F2/Cmnd-2) opens the Browser. The Browser lets you create and examine libraries and classes. If the Browser is already open and in Single Window Mode this option will bring it to the top. If the Browser allows multiple copies of itself, this option opens the initial Browser displaying the libraries. When you open the Browser the Component Store will also open if necessary.

The **Component Store** option (F3/Cmnd-3) opens the Component Store which contains class templates, field and background objects, and external components. If the Component Store is already open this option will bring it to the top.

The **Notation Inspector** option (F4/Cmnd-4) opens the Notation Inspector which lets you view the complete OMNIS notation tree. If the Notation Inspector is already open and in Single Window Mode this option will bring it to the top. If the Notation Inspector allows multiple copies of itself, this option opens a new instance of the Notation Inspector. When you open the Notation Inspector the Property Manager will also open.

The **Inheritance Tree** option (F5/Cmnd-5) opens the Inheritance Tree which lets you view the inheritance or superclass/subclass hierarchy in the current library. If you select a class in the Browser and open the Inheritance Tree it shows the inheritance for that class.

The **Property Manager** option (F6/Cmnd-6) opens the Property Manager which lets you view or change the properties of an object. If the Property Manager is already open this option will bring it to the top.

The **Catalog** option (F9/Cmnd-9) opens the Catalog which lists all the field names and variables in your library, together with the functions, constants and event messages. If the Catalog is already open this option will bring it to the top.

The **Data File Browser** option opens the Data File Browser which lets you create or examine data files. If the Data File Browser is already open this option will bring it to the top.

The **Toolbars** option opens the IDE Toolbar dialog which lets you install or remove the Standard, View, Tools and Desktop toolbars; all of these menus are installed in the top docking at Startup. The toolbars provide some or all of the options in their respective menus. You can drag a toolbar out of its docking area, and drop it onto the left, right, or bottom docking area. You can Right-click on a docking area to show the text for the toolbar(s) currently installed in the docking area.

When you pass the mouse over a particular toolbar button, a tooltip pops up showing you the function of the tool (as shown below on the standard toolbar).

The **Standard** toolbar duplicates the options on the File menu

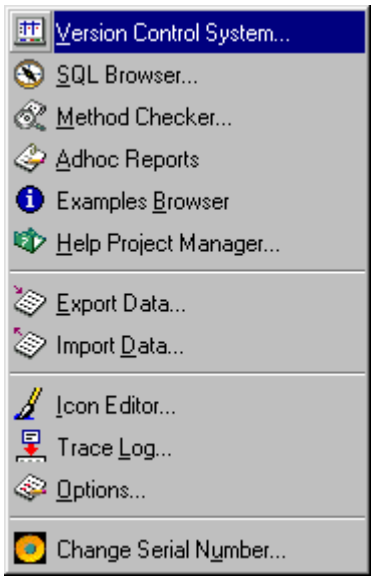
The **View** toolbar duplicates the options on the View menu

The **Tools** toolbar duplicates the options on the Tools menu

The **Desktop** toolbar duplicates the options on the File>>Desktop submenu

The remainder of the View menu shows all the classes you have recently modified. You can select a class name from this list to open the design window for that class. You can also open or instantiate a class from this list, that is, you can open a window, print a report, install a menu or toolbar. For example, to open a window hold down Shift and select the window class name from the View menu.

Tools menu



The **Tools** menu gives you access to various OMNIS development tools.

The **Version Control System** option accesses the OMNIS VCS which enables you to manage your development process. It is described in the *Version Control* chapter.

The **SQL Browser** lets you access and maintain your proprietary databases such as Oracle, Sybase, and Informix. You can also access OMNIS data files using OMNIS SQL via the SQL Object Browser. The SQL Browser is discussed in detail in a later chapter.

The **Method Checker** lets you check your methods for a range of syntactical errors. It is described in the *OMNIS Programming* manual.

The **Adhoc Reports** option installs the Reports menu on the main OMNIS menubar. The Ad hoc report tool lets you generate reports from your server or OMNIS database.

The **Examples Browser** contains a set of examples that demonstrate various OMNIS features. Each example has its own library, which you can examine and re-use. The Libraries tab on the Browser contains all the example libraries, including Extcomp.lbs which shows you external components and Omnispim.lbs which is a demo PIM application.

The **Help Project Manager** lets you build help into your own libraries for the benefit of your own users. It is described under Context Help in the *Extending OMNIS* chapter in the *Omnis Programming* manual.

The **Export Data** option lets you export data from an OMNIS data file using a number of different data formats. The **Import Data** option lets you import data into a data file from an

existing export file or text file from another application. For further details about importing and exporting data see the *Library Tools* chapter.

The **Icon Editor** option opens the Icon Editor which lets you add your own icons to OMNIS. It is described in the *Library Tools* chapter.

The **Trace Log** option displays the trace log which is a record of the operations and commands you have carried out. See the *Debugging Methods* chapter in the *OMNIS Programming* manual for further details.

The **Options/Preferences** menu item displays the OMNIS preferences in the Property Manager. These preferences affect all libraries and OMNIS as a whole, and are described in the next chapter.

The **Change Serial Number** option lets you re-serialize your copy of OMNIS Studio. Serialization is covered in a later chapter.

Window menu

The **Close Top** (Ctrl/Cmnd-W) option closes the top window; the shortcut key is very useful in design mode when you're opening and closing many design windows and method editors. If the current top window is a class design window all changes are saved.

The **Close Other** option closes all windows other than the top one. All changes made to class design windows are saved.

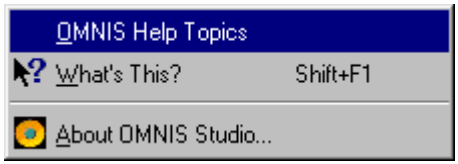
The **Close All** option closes all the windows currently open on the screen. All changes made to class design windows are saved.

The **Print Top** option prints the top window to the current print destination. For example, if the Browser is the top window this option prints the Browser window; if a window design screen is the top window this option prints the window showing its fields and objects.

The Window menu expands to list the first nine windows currently open on your screen. To switch to a particular window, select the corresponding line in the Window menu.

The **Windows** option at the foot of the Window menu opens a dialog containing all the windows currently open on your screen. You can double-click on a window name to bring it to the top.

Getting Help



In design mode, OMNIS provides many different types of help: tooltips and helptips over toolbar controls and the main browsers and editors; help text for the main menus shown in the status bar at the bottom of the OMNIS application window; and a fully context-sensitive Help system. The Help menu gives you access to the latter.

Context-Sensitive Help

The OMNIS Help system contains basic information about how you design applications plus a complete reference for the OMNIS commands, functions, events, properties and methods, and so on.

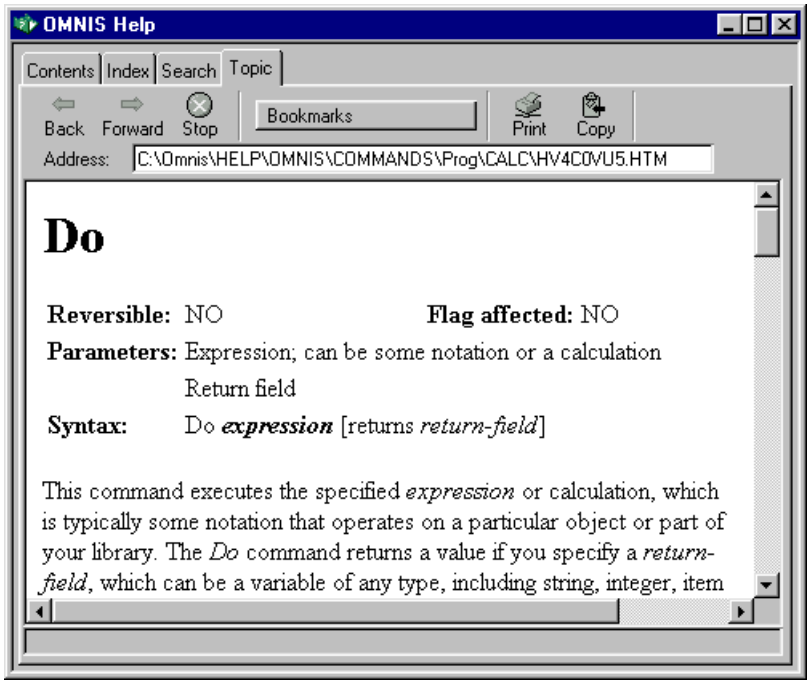
To get help

- Select OMNIS Help Topics from the Help menu

or, to get help for the current editor or selected command

- Press F1 under Windows, or for MacOS either F1 or the Help shortcut key; if the object you select has no help topic, the Search pane is loaded

For example, you can select the *Do* command in the method editor and press F1. The Help window opens at the Topic pane showing help for the *Do* command.



The OMNIS Help window has four panes.

Contents

The Contents pane displays help topics organized by category in a hierarchical list.

- To open or close a book icon, double-click the icon or name, or click on the expand icon
- To view a topic, double-click the topic icon or name

Index

The Index pane displays help topics in alphabetical order. Most objects including commands and functions are listed by topic name.

- To view a topic, double-click the topic name

You can click in the list and type the beginning of a topic name to search for it. OMNIS will highlight the topic which begins with the text you typed. To clear the current search press the backspace key. If the first character is a '*' OMNIS will highlight any topic which contains the text following the *. For example, if you type '*schema', OMNIS will jump to the first topic containing the word 'schema'. You can press the '+' (plus) and '-' (minus) keys to go to the next or previous matching topic.

Search

The Search pane displays a complete list of words contained in the OMNIS Help. If you have a particular word or words in mind this is the place to go.

- To search the list, type the word or words separated by spaces in the entry box at the top of the pane

As you type, OMNIS jumps to the nearest matching word in the word list, and the topics found list shows the topics which contain the word or, for multiple words, *all* of the words you entered.

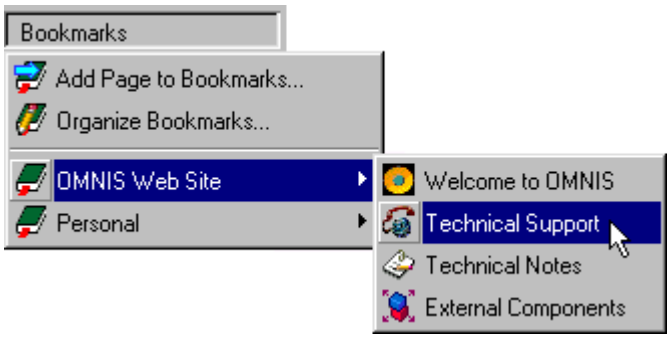
- To view a topic, double-click the topic in the topics found list

Topic

The Topic pane displays the help topic in a Web Browser ActiveX object. Note that if you don't have a suitable Browser object installed on your system, help is displayed in a separate Browser window and the Topic pane is hidden.

The **Back** and **Forward** buttons let you navigate through the topics which you have already visited. The **Print** button lets you print the current topic to the printer. The **Copy** button lets you copy the selected text to the clipboard. Note that some of these buttons are not supported in some versions of the Browser.

The **Bookmarks** menu lets you add, organize, and pick your topics using bookmarks. You can add links to specific help pages in the OMNIS Help, or any page on a website.



The Bookmarks menu contains a few links to the OMNIS website. WARNING: Note that bookmarks to help pages contain the full address of the topic. If you then move your help folder to another location, these addresses will become invalid.

What’s This? Help

OMNIS also provides ‘What’s This?’ help. You can get What’s This? help in a number of ways. For example, you can select What’s This? from the Help menu, or click on the ? button on the title bar of the top window, if it has one, then click on the object.

To get What’s This? help

- Select **What’s This?** from the Help menu

or

- Click on the **What's This?** button on the main OMNIS toolbar

or

- Click on the ? button on the title bar of the top window, if it has one

or you can

- Press Shift-F1 under Windows, or Cmnd-Shift-? under MacOS

A question mark is added to the pointer.

- Click on the object you want help for

The keyboard is disabled while the help cursor is displayed, but you can quit help mode by pressing Escape/Cmnd-period.

Under Windows, the Help menu contains an **About** option which opens the About OMNIS window showing your name, serial number, and the current version number. This option is available on the Apple menu under MacOS.

Shortcut Keys and Mouse Usage

This section lists some useful shortcut keys and mouse usage that you can use to speed up development. Keyboard commands for the standard OMNIS menu options are shown in the menus and will vary across the different platforms.

The usage described below applies to both Windows and MacOS unless indicated. In general, the Ctrl key under Windows corresponds to the Cmnd key under MacOS, while F[number] keys under Windows correspond to Cmnd+[number] under MacOS. For example, F9 corresponds to Cmnd+9, which opens the Catalog.

Note that 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 key and click the mouse under MacOS.

Launching OMNIS & Opening Libraries

Action		
Launch OMNIS	Double-click on the OMNIS icon	
Launch OMNIS and open a library	Double-click on the Library icon	
Open a library (the Browser must be open)	Ctrl/Cmnd-O, or Library>>Open in Browser, or Drop the library’s icon onto the Browser from anywhere in your system	
Open library but *do not* run Startup task	Hold down Alt and open library	Hold down Option and open library
Close the selected library	Library>>Close in Browser	

OMNIS Tools

Action		
Open the Browser	F2	Cmnd-2
Open the Component Store	F3	Cmnd-3
Open the Notation Inspector	F4	Cmnd-4
Open the Inheritance Tree	F5	Cmnd-5
Open the Property Manager	F6	Cmnd-6
Open the Browser Options	F7	Cmnd-7
Open the Catalog	F9	Cmnd-9
Open Find and Replace	Ctrl-F	Cmnd-F

General

Action		
Save the current class	Ctrl+S	Cmnd-S
Open the Print Destination dialog	Shift-Ctrl-P	Shift-Cmnd-P
Print the current selected object (report, method, field, or class); e.g. prints complete list of properties and methods for a class	Ctrl-P	Cmnd-P
Close the top window	Ctrl-W	Cmnd-W
Instantiate a class, e.g. open a window, print a report, install a menu or toolbar	Shift-select class in View menu	
Interrupt current processing, such as method execution, or report printing	Ctrl+Break	Cmnd-. (period)
Exit/Quit OMNIS	Alt+F4	Cmnd-Q

Help

Action		
Get help on currently selected item	F1	
Get ‘What’s This?’ cursor, then click on object to get help	Shift-F1	

Window and Report Design

Action		
Switch between design and open window to t*est* the window	Ctrl+T	Cmnd-T
Make an exact square; or circle; or a line at an angle of 0, 45 or 90 degrees	Ctrl-drag	Cmnd-drag
Duplicate a selected object	Ctrl-drag	Option-drag
Select multiple objects for moving, grouping, locking, or cutting and pasting	Shift-click	
Select all the objects on a window or report	Ctrl+A	Cmnd-A
Lock cursor movement to X/Y direction when sizing or moving object(s)	Shift-drag	
Open the specified class or method editor for the current class	F8	Cmnd-8

Moving and Sizing Objects

Action		
Move object by one pixel, or by grid amount if grid is enabled	←, →, ↑, ↓	
Move object by a greater amount; uses the current grid setting*	Shift-←, →, ↑, ↓	
Size an object by one pixel on its *right* or *bottom* edge	Ctrl+←, → Ctrl+↑, ↓	Cmnd-←, → Cmnd-↑, ↓
Size an object by a larger amount on its *right* or *bottom* edge; uses the current grid setting*	Shift-Ctrl+←, → Shift-Ctrl+↑, ↓	Shift-Cmnd-←, → Shift-Cmnd-↑, ↓

* Uses current window **horzgrid** and **vertgrid** properties regardless of the status of the showgrid, sizetogrid, aligntogrid properties.

List and Grid Fields

Action		
Select multiple non-consecutive lines	Ctrl-click	Cmnd-click
Deselect selected line	Ctrl-click	Cmnd-click
Select multiple consecutive lines	Shift-click	
Deselect all lines (user-defined lists only)	Click on white space at end of list	

The lists key usage applies to most built-in OMNIS lists, as well as your own lists.

Container Fields

Container fields are window fields that contain other fields, such as complex grids.

Action		
To select an object in a selected container field	Ctrl/Cmnd-click on object	
To select all objects in a container field	First select all objects on the window using Ctrl/Cmnd-A, then Ctrl/Cmnd-click inside one of the objects inside the container field	
To drag select a number of objects inside a container field	Ctrl/Cmnd & drag around objects inside the container field	
To copy a container field, including its internal fields	Ctrl/Option-drag on empty part of container field	

Property Manager

Action		
Open droplist or dialog in Property Manager	Ctrl/Cmnd-Down arrow, Return to confirm choice	
To toggle value of property in the Property Manager, e.g. to toggle true/false value or cycle through multiple values	Double-click on value	

Method Editor and Debugger

Action		
Set a go point on a method line	Double-click	
Set a breakpoint	Ctrl+Shift+B	Cmnd-Shift-B
Set a one-time breakpoint	Ctrl+Shift+O	Cmnd-Shift-O
Clear breakpoints	Ctrl+Shift+C	Cmnd-Shift-C
Clear field breakpoints	Ctrl+Shift+F	Cmnd-Shift-F
Stop debugger during execution	Ctrl-Break	Cmnd-period
Display the variable context menu for variable or field	Right-click on variable or field	Ctrl-click on variable or field
Expand calculation/text box in calculation box	Ctrl+U	Cmnd-U
Comment selected method line(s) Uncomment selected method line(s)	Ctrl+; Ctrl+'	Cmnd-; Cmnd-'
Add *new* line below current line in method	Ctrl+N	Cmnd-N
Insert line above current line in method	Ctrl+I	Cmnd-I
Open the window or report design screen for the current method or method(s)	F3	Cmnd-3
Open the method editor for a custom menu installed on menu bar	Shift-select menu option	
Wild card when typing commands, for example, typing o*w finds the *Open window instance* command	* character	
Find next or previous line in list when used with search string, for example, *cur+, +,... finds all strings containing "current"	+ or - keys	

Chapter 3—Libraries and Classes

This chapter introduces OMNIS libraries and describes how you create them. It describes how to create classes, the main components of library files and also includes a description of OMNIS preferences, library preferences and system tables.

Libraries

The main job of application design is to create the objects that define the data structures, data entry windows, reports, pulldown menus, and other user interface elements in your application. These objects are defined as *classes* and are stored in your *library* file.

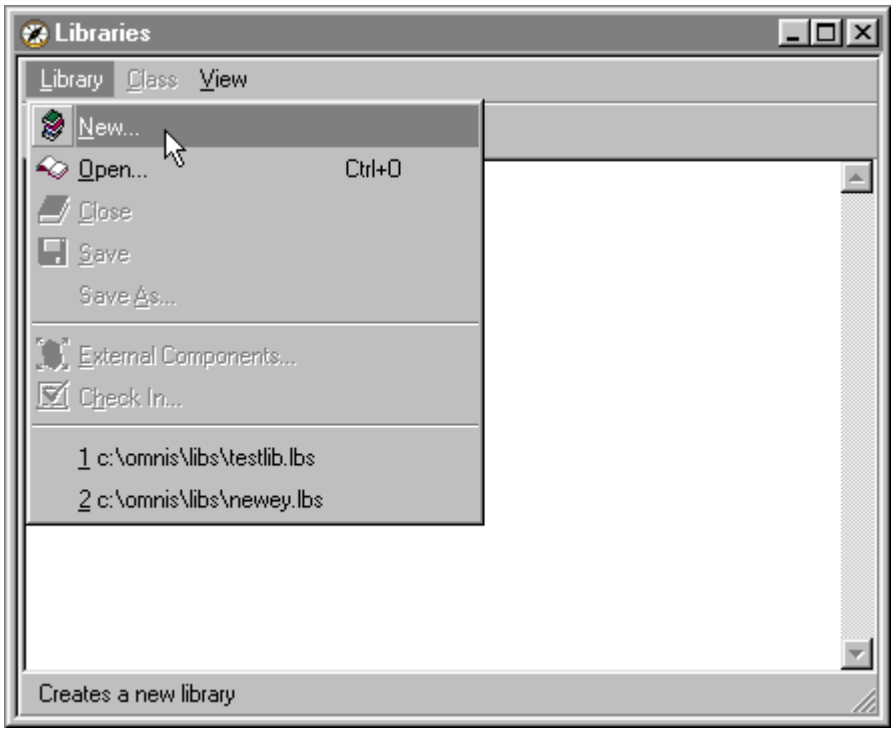
You can create many different types of class in your library. You can create classes either from the Browser or by dragging templates from the Component Store. The Component Store contains a number of templates and wizards that you can use to create classes.

Each library file contains a number of system tables and preferences that control the behavior of your library and its contents. A library has certain properties too, which you can examine and change using the Property Manager.

You can create and open any number of library files and each library can contain any number of classes. In fact you may want to split your whole application into one or more libraries and store different types of objects in different libraries. Alternatively, you can develop your classes, check them into the OMNIS VCS and put them into a library file when you build your final application.

To create a new library

- Start OMNIS and open the Browser from the View menu, or by pressing F2/Cmnd-2
- Select the Library>>New menu option on the Browser menu bar



- Enter a name for the new library in the New Library dialog, including the file extension .LBS, and click on OK

When you name a new library you can use the file naming standards for the current operating system. The .LBS file extension is not obligatory, but it will help you distinguish library files from other types of file. The new library is opened in the Browser and shown as a single icon in icon view, or as a single line in details view.

To open an existing library

- Select the Library>>Open menu option on the Browser menu bar

or

- When the Browser is on top, press Ctrl/Cmnd-O

or if you've opened the library before

- Select the library name from the foot of the Library menu in the Browser

or you can

- Drag a library icon onto the Browser from anywhere in your system

If you try to open a library created with OMNIS 7 Version 3.x, the library will be converted irreversibly. If you want to convert an older library please refer to the *OMNIS Studio Conversion* manual.

To close a library

- Select the library by clicking on its icon or name
- Select the Library>>Close menu option on the Browser menu bar

Library Properties and Preferences

This section describes the properties of a library and the preferences you can set for each library. See also *OMNIS Preferences* at the end of this chapter.

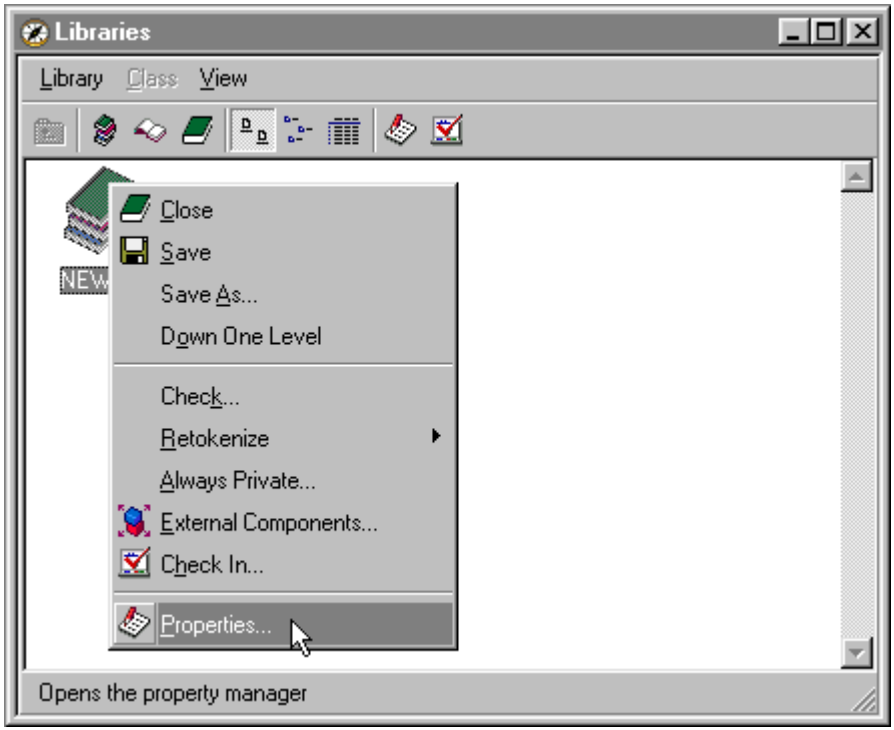
You use the Property Manager to display or change the properties of the current object. In the context of this chapter, you can use it to view and change the properties and preferences of a library.

To view the properties of a library

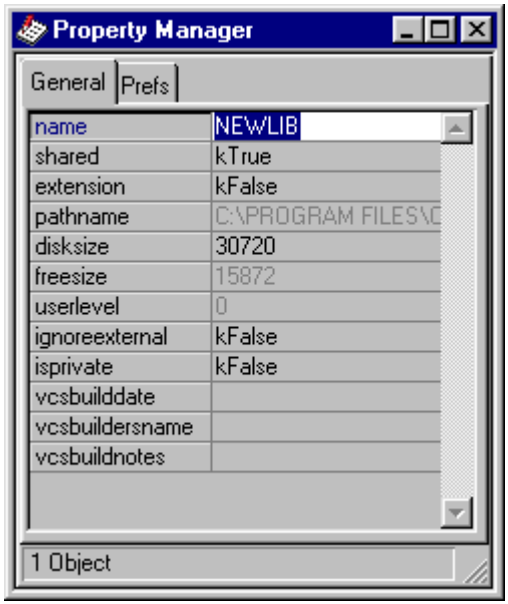
- Open your library in the Browser
- Select the library icon or name
- Select the View>>Properties menu item from the Browser menubar

or you can

- Right-click on the library icon or name in the Browser
- Select the Properties option from the library context menu



The options in the library context menu refer to that particular library. You can use it to close the library, save it, view its contents (Down One Level), Check, Retokenize, or make the library Private: the latter options are described later in this manual. You can also open the External Components Browser and check the library into the OMNIS VCS from the Library context menu; these too are described later in this manual.



The Property Manager opens showing the properties of the current library. These are:

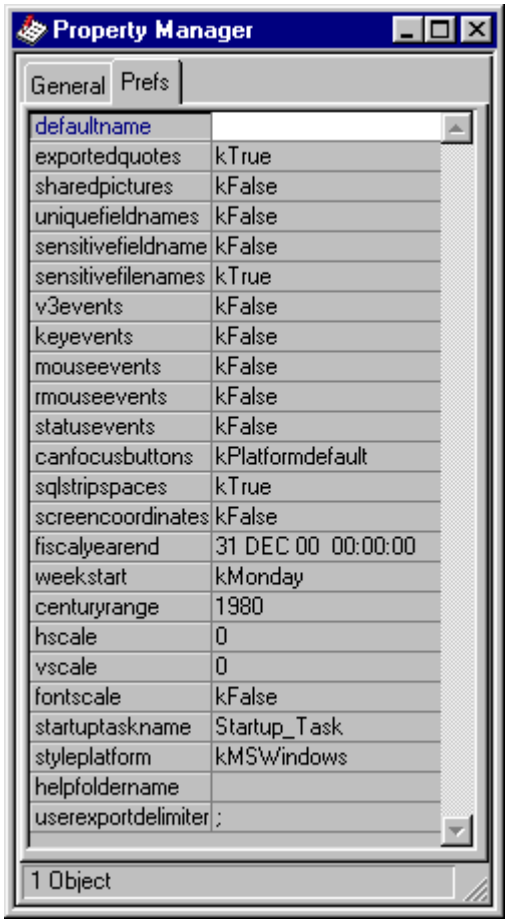
name	the simple name of the library less the .LBS extension
shared	if true the library file is open in shared mode; set this to false if you are the only user
extension	if true the library is an extension library; an extension library placed in the Startup folder is loaded automatically
pathname	the path name of the library file on disk; you cannot change this
disksize	total disk size of the library file, in bytes
freesize	estimated free size in the library file, in bytes; you cannot change this
userlevel	returns the current user number; 0 for master user, or 1 to 8
ignoreexternal	if true you can access a class in another library regardless of the library's external status
isprivate	if true the library is private to the current task
vcsbuilddate	the date the library was last built from the VCS, if applicable
vcsbuildersname	the person who last built the library from the VCS, if applicable
vcsbuildnotes	any notes recorded when the library was built from the VCS, if applicable

To view the preferences for a library

- Open your library in the Browser
- Right-click on the library icon or name in the Browser
- Select the Properties option from the library context menu
- Select the Prefs tab in the Property Manager

or

- Select your library in the Browser and Press F6/Cmnd-6
- Click on the Prefs tab in the Property Manager



The library preferences are stored in the library file and control the behavior of that particular library and its contents. The preferences are summarized here.

defaultname	the default name of the library; you can reference classes outside the current library using defaultname.classname
exportedquotes	if true text is exported in quotes
sharedpictures	if true pictures are converted to OMNIS shared format; they are then visible under all platforms
uniquefieldnames	if true file class field names must be unique, that is, fields in different file classes are not allowed the same name
sensitivefieldnames	if false the case of file class field names is ignored; otherwise they are case sensitive
sensitivefilenames	if false the case of file class names is ignored; otherwise they are case sensitive
v3events	true if v3 events are enabled, true for converted v3 libraries, false for new libraries; unless you are running a converted library you should leave this set to false
keyevents	activates events for key presses
mouseevents	activates events for mouse clicks
rmouseevents	activates events for right-button mouse clicks
statusevents	activates events for field status, that is, events when the field is enabled, disabled, shown, or hidden
canfocusbuttons	if true buttons, check boxes, radio buttons get the focus on open windows
sqlstripspaces	if true trailing spaces are stripped from retrieved character columns when data is fetched from SQL
screencoordinates	if true hscale and vscale properties scale open windows; this also enables fontscale
fiscalyearend	the date of the fiscal year end; used in financial date calculations
weekstart	the starting day of the week for date calculations
centuryrange	the starting year of the century range you are using; defaults to 1980
hscale	horizontal scale factor of the main OMNIS window when screencoordinates property is set
vscale	vertical scale factor of the main OMNIS window when screencoordinates property is set
fontscale	if true font scaling occurs when screencoordinates is set
startuptaskname	name of the task class that opens when the library opens; defaults to “Startup_Task” for new libraries

styleplatform	specifies the current platform for window field styles
helpfoldername	the name of the help folder for your library, located in the HELP folder under the main OMNIS folder
userexportdelimiter	the field delimiter to be used on export, default is ;

Initially you won't need to change any of these library preferences. However, if you are developing a client/server application you may need to change the **uniquefieldnames** property. For further details see the *OMNIS Programming* manual.

Default Library Name

The **defaultname** property stores the internal name for a library used throughout your application to reference classes. The defaultname property defaults to the disk file name of your library, but you can assign any name you want to the property. If you rename the library file on disk the defaultname remains the same retaining all class references. If you are using multiple libraries you should set defaultname for all your open libraries and use it to refer to classes outside the current library. If you change the defaultname property after you start developing your library, all class references that use it will fail; therefore in a multi-library system you should set it once before you start adding classes to your library.

Classes

Classes are predefined structures that control the appearance and behavior of the objects in your application. Classes are the main components in your library file. You can create classes from the templates provided in the Component Store, or you can create them from the Browser. You can create any number of classes in each library file and modify them at any time while you develop your application.

There are several different types of class in OMNIS, each one performing a particular function in your library, or in your application as a whole. In general, classes are either *Data classes*, *GUI classes*, or *Container classes* depending on the type of object they define and what other objects they contain. The types of class are

Schema
data class that defines a server table and its columns on your server database

Query
data class that defines one or more server tables and their columns on your server database

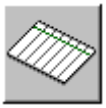


Table
data class that maps to a schema class and contains methods for processing your server data

File
data class that defines the structure in an OMNIS data file

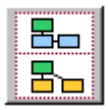
Search
class that filters data stored in an OMNIS data file

Window
gui class that defines the data entry windows and dialogs in your application

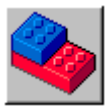
Menu
gui class that defines standard pulldown, popup, and hierarchical menus in your application

Toolbar
gui class that defines the toolbars in your application

Report
gui class that defines the reports you can print in your application



Task
class that contains or controls other instances, and handles events in your application



Object
class that contains methods and variables defining your own structured data objects

Code
class that contains global methods you can use throughout your application

This chapter describes the general characteristics of classes and how you create them from the Component Store. Individual classes are discussed in separate chapters in this manual.

To create a new class from the Component Store

You can create a new class using drag and drop when the Browser is showing the classes in your library. In this case you can drop the new class anywhere on the Browser. To do this

- Show the classes in your library in the Browser, using View>>Down One Level
- Drag the type of class you want to create from the Component Store onto the Browser

or if the Browser is displaying libraries you can

- Drag the type of class you want to create from the Component Store onto your library icon or name in the Browser
- Release the mouse when the Browser or library highlights

To create a new class from the Browser

- Show the classes in your library in the Browser, using View>>Down One Level
- Select the Class>>New option in the Browser menu bar
- Select the type of class you want to create

If one of the options under Class>>New menu is grayed out, that type of class is not currently visible in the Browser and OMNIS will not let you create that type of class from the Component Store or the Browser. If you want to create classes of this type you need to show them using the Browser Options available under the View menu or press F7/Cmnd-7.

When you create a new class in the Browser, OMNIS gives it a default name, normally "New <Class>", that is, the name of the default template for that type of class. You can accept this name or change it to something more appropriate to your library.

To name or rename a class

Assuming you have created a class in the Browser

- Click on the class to highlight its name
- You may need to click on the name again to make the name enterable
- Type the new name of the class
- If you delete the name altogether, you can press Esc to get back to the original name

If you rename a class, OMNIS will ask you if you want to find references to the old name and replace them with the new class name. If you answer Yes, the Find and Replace dialog is opened which lets you change all references to the class throughout your library.

OMNIS does not impose any restriction on the characters you use to name a class. However, you should avoid using all non-alphanumeric characters including all punctuation marks such as commas, colons, periods, forward and backslashes, and all types of brackets. Furthermore, you should avoid adding leading or trailing spaces to class names.

To copy a class from one library to another

- Open each library in a separate Browser window
- View the classes in the library containing the class you want to copy, using the View>>Down One Level option in the Browser menu bar
- Select the class or any number of classes and drag your selection onto the destination library

or

- Select the class or classes and use Edit>>Copy to copy your selection to the clipboard
- Select your destination library and use Edit>>Paste to paste the class or classes

Note that using the clipboard lets you copy classes between libraries when the Browser is in single window mode. When you copy classes from one library to another you may need to copy other associated files. For example, if the window classes in your library use field styles you must copy the #STYLES system table containing the field styles into your destination library, and you should copy across the system table first before any window classes. For complete library and component management you should use the OMNIS Version Control system.

Printing Classes

When you print a window or report class from the Browser, you get a summary of the objects on the window or report. If you want to print detailed information about one or more objects on a report or window class, you need to open the window or report class editor, select the object or objects you want to print, and use the print button on the standard IDE toolbar or the print item on the File menu.

You can print a summary list of the classes currently displayed.

To print the summary list to the current print destination

- Deselect all classes

Then

- Press the print button on the standard IDE toolbar

or

- Select the Print Class Details item on the standard File menu

or

- Press Ctrl-Cmnd-P

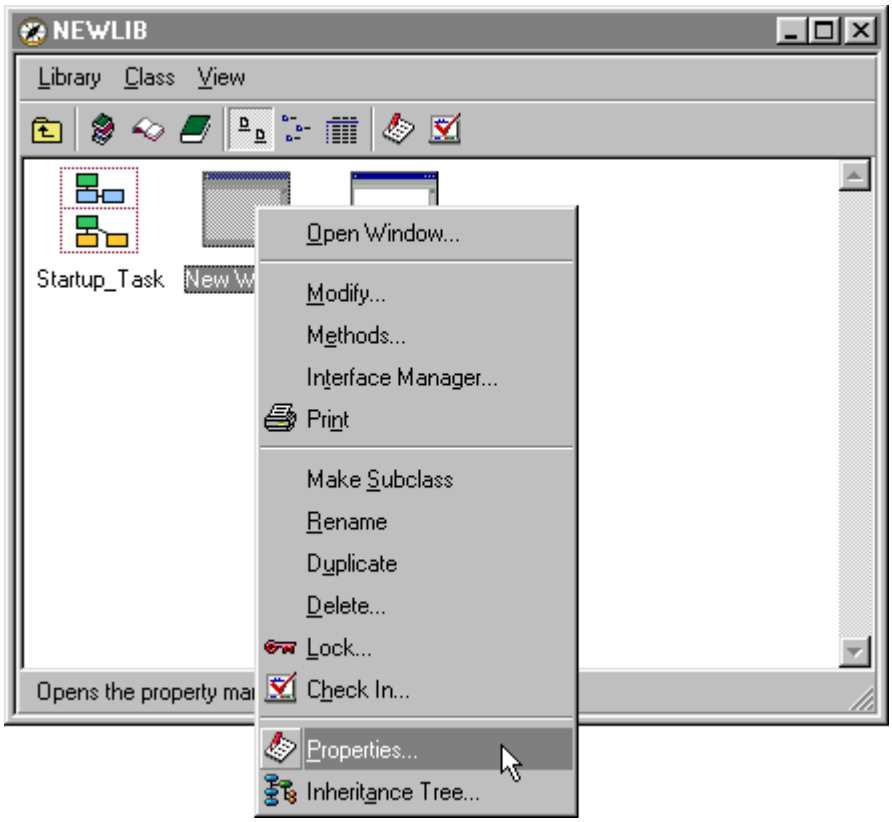
Locked classes and system table classes cannot be printed.

Class Properties

All classes have certain general properties, as well as their own special properties. For example, they all have a name, a type, creation date, and so on. This section describes the common class properties, whereas subsequent chapters describe class-specific properties, including those to do with inheritance, when you need to know about them.

To view the properties of a class

- View the class in the Browser (any view will do, Icons or Details)
- Right-click on the class and select the Properties menu item

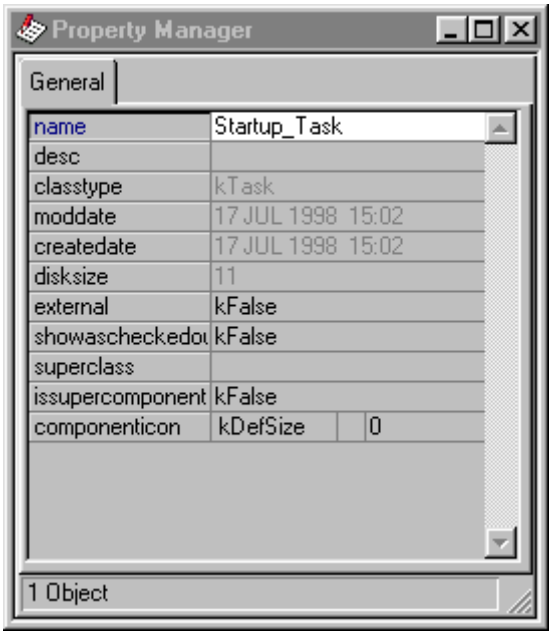


or

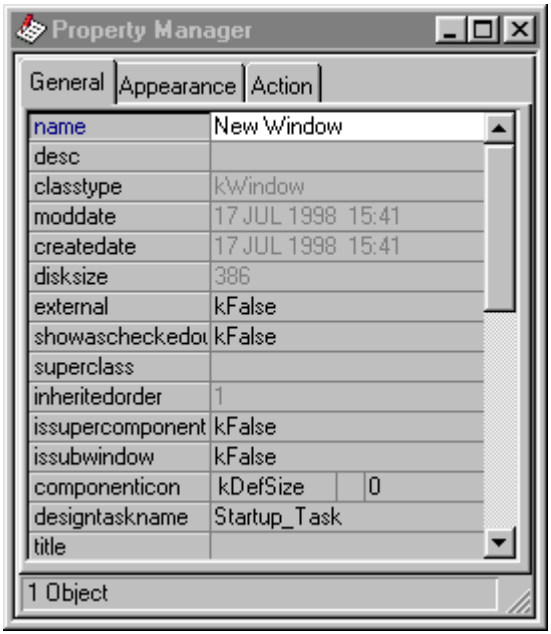
- Select the class in the Browser and select View>>Properties from the Browser menubar, or press F6/Cmnd-6

or for some classes, such as windows and reports, you can

- Double-click on the class to open it in design mode
- Right-click on the background of the class design screen, and select the Properties option from its context menu



The properties of a task class



The properties of a window class

As shown, some types of class, such as task classes, have only a few common properties while others, such as window classes, have many other properties that are unique to the class. In the Property Manager you can change the properties of a class, but you can't change a property that is shown in red or where its value is grayed out.

Most of the common or general properties of a class are self-explanatory. The **external** property means the class is visible to other libraries, normally set to false. The **designtaskname** is the task in which you expect to use the class, and it enables you to use the task variables of that task in the class. The **showascheckedout** property shows whether the class has been checked out of the OMNIS Version Control System. See the *Version Control* chapter for more information.

Note that you *cannot* change the **classtype, moddate, createdate,** or **disksize** properties in the Property Manager, but you can change the class name or add a description, although a description is not essential.

The class context menu, which you get by right-clicking on the class in the Browser or in the class design screen itself, also lets you

- **Open** or **Print** the class
  open an instance of the class, e.g. open a window, install a menu, print a report
- **Modify** the class
  that is, open the design window for the class
- **Methods**
  opens the method editor for the class which lets add methods and variables to the class; this option is grayed for classes that cannot contain methods
- **Interface Manager**
  opens the Interface Manager for the class which displays the public methods and properties of the class
- **Print**
  sends a report of the properties, variables and methods of the class to the currently selected print destination
- **Make Subclass**
  creates a subclass of the selected class
- **Rename** the class
  when you rename a class all references to the class are changed
- **Duplicate** the class
  creates an exact copy of the class with the name "<Class> copy"
- **Delete** the class
  the class is permanently removed from the library file; *this is irreversible*
- **Lock** the class
  when you lock a class you can no longer view or modify it; *this is irreversible*
- **Check In**
  checks the class into the VCS; this is only available if you have the OMNIS VCS
- **Properties**
  shows the properties of the class
- **Inheritance Tree**
  opens the Inheritance Tree with the current class selected

Some of these options are duplicated in the Class menu in the Browser, and act on the selected class or a number of selected classes.

Default Classes

When you create a new library in OMNIS, it contains certain default classes including a task class called *Startup_Task* and various *system tables* that control the appearance of your library. To start creating your own classes in your library you don't need to do anything with these default classes, but this section gives you a brief overview of how they affect your library. For full details about using tasks, see the *OMNIS Programming* manual.

The Startup Task

When you create a new library it contains a task class called Startup_Task. When you open your library the startup task is opened and the initialization code within it is run automatically. Therefore if you want your library to do something in particular when it starts up, you put the code to do it in the startup task. For example, you might want your library to open an About screen or install a menu. You would put the code to do these things in your start up task.

Each library has a preference called **startuptaskname** which stores the name of the startup task, and is set to Startup_Task by default. To change the task that is run when your library opens, you need to change this property, but in most cases you can leave it set to Startup_Task.

The startup task has a special function when you are designing your library and adding other classes and variables to your library. At present you don't need to worry about the startup task or do anything to it, you can proceed to create your data and gui classes in your library.

System Tables

Every new library contains a number of *system tables*. However the default options in the Browser mean they are hidden when you create and view a library. You can show them by changing the Browser options, or by pressing Shift-Ctrl/Cmnd-A to show all the classes in your library. The system tables are prefixed with "#".

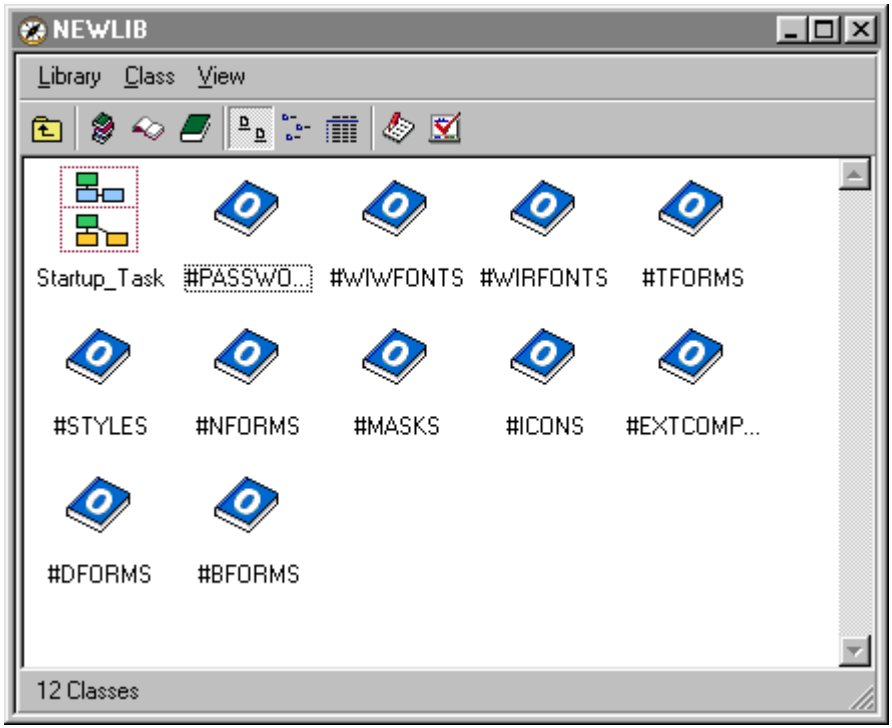
System tables are special types of class that hold information about the OMNIS environment, including field styles, fonts, input masks, and external components. You can edit some of the system tables to change the way OMNIS behaves. The settings for these tables are stored for each separate library. You can copy system tables from one library to another and you can edit them, but some options available for normal classes are not available for these tables.

To view the system tables for a library

- Open your library in the Browser and display its classes
- Select View>>Browser Options on the Browser menu bar, or press F7/Cmnd-7
- Check the System Tables option and click OK

or when the Browser is the top window

- Press Shift-Ctrl/Cmnd-A to show all classes, including the system tables



Most of the system tables are described in details in the *OMNIS Programming* manual; they are summarized here.

Name	Description
#BFORMS	boolean formats; these specify the format of Boolean fields allowed in your library
#DFORMS	date formats; these specify the format of short date, and date and time values
#EXTCOMPLIBS	the external components available in the current library: described in the *External Components* chapter
#ICONS	the icon file for the current library; described in the *Library Tools* chapter
#MASKS	input masks for data entry fields
#NFORMS	number formats for numeric data entry fields
#PASSWORDS	the master and user passwords for your library; described in the *Library Tools* chapter
#STYLES	character styles for window and report fields, and text objects
#TFORMS	text formats; these specify the format of character fields
#WIRFONTS #MARFONTS	font table for *report* classes under Windows or MacOS; in this system table you can map fonts used on report classes under one operating system to fonts appropriate for the other OS
#WIWFONTS #MAWFONTS	font table for *window* classes under Windows or MacOS; in this system table you can map fonts used on window classes under one operating system to fonts appropriate for the other OS

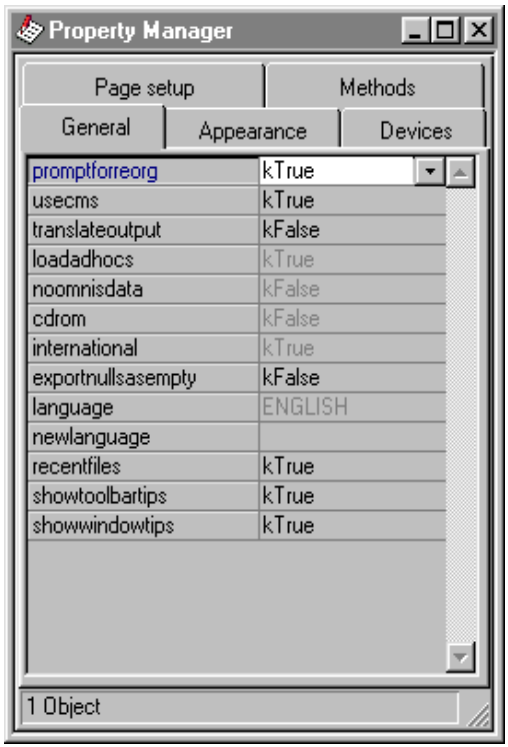
You can edit a system table by double-clicking on it in the Browser. For example, you can double-click on #DFORMS which opens the Date formats dialog showing all the date formats for the current library.

OMNIS Preferences

This section describes the global preferences or options in OMNIS, which are the properties of OMNIS itself. You use the *Property Manager* to display or change them. Most of these preferences affect how your libraries behave.

To view the OMNIS preferences

- Select the Tools>>Options menu item from the main menu bar; this option is called Preferences under MacOS



The Property Manager opens showing the OMNIS preferences, organized on several panes. These options control the overall behavior of OMNIS. You can change any of the preferences shown in blue, but not those shown in red; note that you can change the default color of properties you can and can't set in these preferences. The preferences are as follows.

General Preferences

promptforreorg	if true OMNIS prompts you when a data file needs reorganizing
usecms	if true centimeters are used for report measurements: when false inches are used as the default units
translateoutput	when true OMNIS converts the OEM character set to ANSI when importing or exporting data to a file, port, or the clipboard: only has affect for character values greater than 127
loadadhocs	true if your serial number allows access to ad hocs
noomnisdata	true if your serial number does not allow access to data files
cdrom	true if your serial number does not allow write-access to data files and libraries, that is, they are accessed from a CD-ROM
exportnullsasempty	when true NULL values are exported as empty values
language	the current language as defined in OMNISLOC.df1
newlanguage	the language used when you restart as defined in OMNISLOC.df1
recentfiles	if true recent open libraries are added to the Browser Library menu, recent open data files to the Datafile menu in the Data File Browser, and recent icon datafiles to the icon editor menu
designscreenaid	dotted area on the OMNIS development screen to help you design for smaller screen sizes (MacOS only)
stickywindowmenubar	(MacOS only) If true, menus in OMNIS windows behave in the MacOS 8 style; otherwise behavior reverts to pre-OS 8 style
showtoolbartips	if true toolbar tooltips are shown in OMNIS and your libraries
showwindowtips	if true window object tooltips are shown in OMNIS and your libraries

Appearance Preferences

helpbaron	if true shows the help status bar at the foot of the OMNIS window on the PC, or at the foot of the monitor on Mac.
balloonson	if true help balloons are turned on (MacOS only)
helpfont	sets the font type and size of the help status bar and balloons
inheritedcolor	color of inherited objects and properties
setpropertycolor	color for properties that you *can* set in the Property Manager
nosetpropertycolor	color for properties that you *cannot* set in the Property Manager
toolobjselectcolor	color for the currently selected control in toolbar design mode
runtimepropertycolor	color for runtime properties shown in the Property Manager

style97	if true toolbar and icon array buttons are flat; you must restart OMNIS for this option to take effect
taskbar	displays a taskbar at the top or bottom of the OMNIS application window, default is none (Win 95 & NT only)
minimizeiconid	the iconid for the OMNIS windows when it is minimized
fullscreen	if true OMNIS is running in fullscreen view; the OMNIS window is maximized, and the main menu bar is removed

Devices Preferences

See the *Report Classes* chapter for more information.

reportfile	the full path and file name for the Disk device
printfile	the full path and file name for the File device
editionfile	the full path and file name for the DDE/Publisher device
pages	the page or page numbers to be sent to the device
reportdataname	the name of the binary field for the Memory device
reportfield	the name of the window field for a Preview or Screen report
windowprefs	the optional title and screen co-ordinates for a Screen or Preview window
waitforuser	if true, method execution is halted until the user closes the Screen or Preview window
hideuntilcomplete	if true, a Screen or Preview window remains hidden until the report is finished
c**harsperinch**	the number of characters per inch when printing to a text-based device
linesperinch	the number of lines per inch when printing to a text-based device
generatepages	if true, reports generate paged output when printing to text-based devices, that is, page headers and footers are generated as normal; if false, only one report header and page header is printed at the beginning of the report
linesperpage	the number of lines per page when $generatepages is true
restrictpagewidth	if true, the width of a page is restricted when printing to text-based devices
charsperline	the number of characters per line when $restrictpagewidth is true
sendformfeed	if true, form feeds are sent to text-based devices after each page
appendfile	if true, data is appended to the current print file specified in $printfile, if false, the file is overwritten when printing to the File device

istext	if true, forces a non-text device to behave like a text-based device using the same preferences as text-based devices
portname	the name of the port when printing to the Port device
portspeed	the port speed setting when printing to the Port device
porthandshake	the handshake when printing to the Port device
portparity	the parity checking when printing to the Port device
portdatabits	the number of databits when printing to the Port device
portstopbits	the number of stop bits to be used when printing to the Port device

Page Setup Preferences

See the *Report Classes* chapter for more information.

pagesetupdata	stores the page setup data for the class. You can click on the down arrow to open the page setup dialog. When the page setup is specified the property displays (Not empty); you can delete this text to clear the current page setup
orientation	the page orientation;
paper	the paper type constant
paperlength	the length of the paper in cms or inches depending on the usecms preference
paperwidth	the width of the paper in cms or inches depending on the usecms preference
scale	the scaling factor in percent
copies	the number of copies

Methods

serialize	when executed, this method opens the OMNIS serialization dialog; see the Serialization and Installation chapter

Chapter 4—Variables and Methods

This chapter describes how you add variables and methods to the objects in your library using the *method editor*.

In OMNIS the principal data container is the *variable*. Most objects can contain variables, but their scope and the kind of data they can contain depends on the *type* of variable. You can add variables to any class that can be opened or instantiated, or any class that can contain methods, namely window, menu, toolbar, report, task, table, code, and object classes. You use the method editor to add variables to the objects in your library.

Methods are pieces of code contained in a class or object that perform some action when the object receives the appropriate message. In OMNIS every object has certain default or standard methods, but you can add your own methods to either override the standard behavior or add functionality to the object. You can add methods to do everything from handling events and controlling program flow, to sending SQL statements to a server, to printing statements from an invoices application. Like variables, you can add methods to any class that can be opened or instantiated, and you use the method editor to do this.

Furthermore, you can add methods behind window and report fields, menu lines, and toolbar controls. You use the method editor to add and debug methods for these objects too.

Data Types

Before you can add variables to the objects in your library you need to understand the different *data types* available in OMNIS. Choosing the right type for your data ensures that OMNIS will do the right thing in computations requiring conversion. It also lets OMNIS validate the data as you enter or retrieve it.

This section describes in detail the standard data types you can use to represent data in OMNIS. Some of the basic data types have subtypes, or restrictions of size or other characteristics of the data that give you finer control over the kind of data you can handle. The following data types are available.

Character	standard character set sorted by ASCII value
National	standard character set sorted by national sort order
Number	multiple types for representing integers, fixed point and floating point numbers
Boolean	single-byte values representing true or false values, or their equivalents
Date Time	multiple types for representing simple dates and times, or composite date and times, between 1900 and 2099 to the nearest hundredth of a second
Sequence	proprietary data type for numbering OMNIS data file records
Picture	stores color graphics of unlimited size and bit-depth in platform-specific format or in a proprietary shared picture format
List	structured data type that holds multiple columns and rows of data of any type
Row	structured data type that holds multiple columns of data in a single row
Object	your own structured data type based on an object class
Binary	stores any type of data in binary form, including BLOBs
Item Reference	stores the full notation of an object in your library or OMNIS itself
Field Reference	passes a reference to a field (parameter variables only)

Character

Character data can contain characters from any of the various single-byte standard character sets. You can define a Character column of up to 10 million (10,000,000) bytes in length. Character columns or fields generally correspond to SQL VARCHAR data and have a varying length format.

In OMNIS character data is sorted according to its ASCII character set representation, not the server representation. The ASCII character set sorts any upper case letter in front of any lower case letter. For example, these character values

```
adder, BABOON, aSP, AARDVARK, Antelope, ANT
```

are sorted as

```
AARDVARK, ANT, Antelope, BABOON, aSP, adder
```

National

Like Character data, National data can contain characters from any of the various single-byte standard character sets. You can define a National column of up to 10 million (10,000,000) bytes in length. However, when you sort National data, OMNIS sorts the values according to the ordering used by a particular national character set.

The ordering for the English language follows: A, a, B, b, C, c, D, and so on. For example, if the previous values were values of a national column or field, OMNIS would sort them as follows:

```
AARDVARK, ANT, Antelope, adder, aSP, BABOON.
```

If you store data in an OMNIS data file, OMNIS stores a copy of the ordering in the file along with the data. If you use the data file on another machine, OMNIS preserves the original ordering.

Number

A *number* is an integral or floating point number having various storage and value characteristics, depending on its subtype. The following table summarizes the different subtypes for numbers.

Number type (dp = "decimal places")	Storage (bytes)	Range
Short integer	1	0 to 255
Long integer	4	-2,000,000,000 to +2,000,000,000
Short 0 dp	4	-999,999,999 to +999,999,999
Short 2 dp	4	-9,999,999.99 to +9,999,999.99
Floating dp	8	approx -1E100 to +1E100 (16 significant digits)
Number 0 dp	8	-999,999,999,999,999 to +999,999,999,999,999
Number 1 dp	8	-99,999,999,999,999.9 to +99,999,999,999,999.9
Number 2 dp	8	-9,999,999,999,999.99 to +9,999,999,999,999.99
Number 3 dp	8	-999,999,999,999.999 to +999,999,999,999.999
Number 4 dp	8	-99,999,999,999.9999 to +99,999,999,999.9999
Number 5 dp	8	-9,999,999,999.99999 to +9,999,999,999.99999
Number 6 dp	8	-999,999,999.999,999 to +999,999,999.999,999
Number 8 dp	8	-9,999,999.999,999,99 to +9,999,999.999,999,99
Number 10 dp	8	-99,999.999,999,999,9 to +99,999.999,999,999,9
Number 12 dp	8	-999.999,999,999,999 to +999.999,999,999,999
Number 14 dp	8	-9.999,999,999,999,99 to +9.999,999,999,999,99

Floating Point Numbers

There are many pitfalls in using floating point numbers in programming. Computers do not represent these numbers exactly, only approximately within the precision of the machine. This can lead to all kinds of anomalous problems with comparison of values, particularly with equality comparisons. Two floating point numbers may differ by an infinitesimal amount depending on the technique used to generate the values, even though logically they should be the same.

In general, you should not use equality comparisons with floating point numbers. If you are working with “fixed-point” data such as money values, use scaled integers for comparison and arithmetic.

For example, instead of comparing two floating point variables F1 and F2 containing the amounts $5.00 and $10.00, compare two integer variables I1 and I2 containing 500 and

1000. Display I1 * .01 when you need a decimal value. You can also use the *rnd*() function to round the numbers to a certain number of decimal places before comparing them.

Boolean

The *boolean* data type represents single-byte values of true (yes), false (no), empty, or null. You should take care to give each Boolean column or field an initial value, because OMNIS initializes boolean data to “empty”, not NO or null.

When used in a data entry field in a window, boolean data is treated as three characters in which any data entry is interpreted as a YES or NO. A `Y', 'YE' or 1 is seen as YES while an 'N' or 0 will suffice for No. If the field is a check box, you enter the boolean value by clicking on the box. If you don't initialize the field and the user does not click on the box, the field has an “empty” value.

You can use boolean values in expressions. The numeric value is 1 for Yes values and 0 for No and empty values. NULL values are treated as undefined in numeric calculations. For example, (null+1) is null and (null>1) is null.

When converted to character strings, Boolean columns or fields can take values "YES", "NO", "NULL", or empty, "". In some cases, for example when setting up search criteria, you can enter values other than these for a Boolean field; in this case, OMNIS converts them and matches them with empty. Thus, for example, the value 'FALSE' is converted to empty, as are values like SAM, HAPPY, and so on.

Date Time

The date and time group of data types contains three basic subtypes: a four-byte *Short date*, a two-byte *Short time*, and an eight-byte *Long date and time*. The following table summarizes the date and time subtypes.

Date Time subtypes	**Storage (bytes)**	**Range**
Short date	4	1900..1999
Short date	4	1980..2079
Short date	4	2000..2099
Short time	2	Minute resolution
Date time(#FDT)	8	Formatted #FDT, to centiseconds
Date time(D m Y)	8	Formatted D m Y, to centiseconds

Note that the display of dates depends on the settings in the #DFORMS system table. Also the long date and time subtypes are identical in value, only displaying differently in window fields.

Short Date

The *short date* type spans the range JAN/1/0000 to DEC/31/9999. There are three specific built in ranges: 1900 to 1999, 1980 to 2079, and 2000 to 2099. By choosing the appropriate range, you can enter just two digits for the year and OMNIS recognizes the century correctly. For example, if you select the range as 2000 to 2099, a date you enter as 7,12,57 will be read as 7,12,2057 rather than 1957. To enter a date outside the three specific year ranges, you need to set up your own date display format.

Using these date ranges you will avoid the Year 2000 problems inherent in some other systems.

OMNIS accepts dates in different formats automatically, with the exact format depending on whether your system is US or European. For example, you could enter the 7th of December, 1998, as any of the following strings.

US system	European system
12-7-98	07-12-98
12/7/98	7/12/98
12%7%98	7%12%98
DEC 7 98	7 DEC 98

The US and UK versions of OMNIS have different date displays because their default date format strings are different.

You can use any character to delimit the day and month figures. If you don't specify the year or month and year, OMNIS assumes the current year or month and year, respectively.

OMNIS supports three kinds of date arithmetic in expressions.

– Addition of days:
  Date + Days = Date (forward)

– Subtraction of days:
  Date - Days = Date (back)

– Subtraction of dates to yield number of days between the dates:
  Date1 - Date2 = Number of Days between the dates

OMNIS uses the string variable #FD to define the display format of dates. There are also several date functions that let you manipulate date strings.

Short Time

Short time types have two-byte values in the form HH:NN. The range of possible time values is from 00:00 to 23:59.

You can use time in expressions. OMNIS automatically converts the time into numeric values using the conversion HH*60+NN, giving the total number of minutes. The #FT string variable controls conversions between time and string types.

Long Date and Time

The combined *Date Time* type can hold a complete date and time to 1/100th second. It has various subtypes depending on the display format you select (stored in #FDT) and uses 8 bytes when stored in a data file.

Date and Time Calculations

The numeric value of a date or time variable in an expression depends on the format string for that variable. So, if DATE1 has date format string H:N and DATE2 has date format string H:N:S, DATE1 has a numeric value equal to the number of minutes since midnight and DATE2 has numeric value equal to the number of seconds since midnight. It follows that DATE1+1 adds 1 minute to DATE1 and DATE2+60 adds 1 minute to DATE2.

Addition and subtraction involving two date/times cause the numeric value of each to adjust so that they are both based on a common denominator. Thus DATE1-DATE2 returns a numeric value equal to the correct difference between the two times in seconds. However, DATE1*1-DATE2*1 loses the information that DATE1 and DATE2 represent date times and returns a meaningless difference between the DATE1 value in minutes and the DATE2 value in seconds, for example, 500 minutes - 600 seconds.

Note that calculations involving combined dates and times do not work properly if the date part is before 1900. Comparisons between two datetimes with different date format strings work properly.

When you compare parts of dates, for example, the month part of a date, dtm('11 June 98'), OMNIS compares the string representation of the month unless some calculation forces it to use the number representation of the month. Thus the expression dtm('11 Dec 98') is less than dtm('11 June 98') because 'D' is before 'J' in the alphabet. To force a correct numeric comparison, add 0. For example

```
If dtm('11 June 98')<(dtm('11 Dec 98')+0)
   OK message {6 is less than 12}
End If
```

You should try to use straight date comparisons if you are comparing full dates. Don’t try to convert them into integers or other types of data. Let OMNIS do the work for you.

Century Ranges for Dates

When entering data into a date time field or variable without specifying the century, the date normally defaults to be within the hundred year range starting with 1st January 1980. However, you can specify the start of the hundred year default range as a library preference with the option of overriding it for individual date types.

You can use the $centuryrange library preference to set the default century range ($clib.$prefs.$centuryrange), a four digit year is specified which defaults to 1980. So if, for example, $centuryrange is set to 1998, dates for which no century is entered default to between 1st January 1998 and 31st December 2097.

In addition, the 30 date formats which are stored in the #DFORMS system table can include the century range by including a four digit year at the end of the date format. For example, date formats starting at 1st January 1998 include ‘D m Y H:N:S 1998’ and ‘YMD 1998’. This can be used to override $centuryrange for particular date types. The same mechanism can be used to control the conversion of character values to dates using the dat() function, for example:

```
Do dat(charvar,’D m Y 1998’) Returns datavar
```

Century ranges are used when dates are entered from the keyboard or when a character string is converted to a date. If you enter a date that includes the century, the century range is ignored. Century ranges do not affect how a date value is stored or displayed, OMNIS always stores the full date including the century.

Sequence

Every time OMNIS inserts a new record into an OMNIS data file, it assigns a unique number, a *record sequencing number* or RSN to that record. There is a special data type, the *sequence*, for this type of data. Each RSN references a location in the data file. If you delete a record, OMNIS does not reuse the RSN. The RSN is stored as a 32-bit integer so its maximum value is 2^32-1, which is approximately 4,295 million! The sequence type is not applicable to client/server data.

OMNIS assigns record sequencing numbers (RSNs) according to the following rules:

– The first record in a file has RSN 1, the second record RSN 2, and so on

– An RSN is never used again, even though the record may no longer exist

A window field with sequence type provides a way for the user to see the RSN for any record in an OMNIS data file, even though they cannot change it.

OMNIS assigns the RSN just before saving the record in the data file, so it is not available for any calculations prior to the *Update files* command.

Picture

The *picture* data type holds color graphics with a size limited only by memory. The space each picture consumes depends on the size and resolution of the image. The internal storage of a picture is either in native format (Windows bitmap or DIB or metafile or Macintosh PICT) or in OMNIS shared color format. Server databases store picture data as binary objects (BLOBs).

List

The *list* is a structured data type that can hold multiple columns and rows of data. A list can hold an unlimited number of lines and can have up to 400 columns. When you create a list variable you set the type of each column. The data type of each column in your list can be any one of the other data types including Character, Number, Date, Picture, and List: Yes, you can even have lists within lists!

OMNIS makes use of the list data type in many different kinds of programming tasks. Normally you would create a variable with list data type and build your list in memory from your server data or OMNIS data file. Then you could use your list data as the basis for a grid or list field on a window, or you could use it to generate a report.

You can store lists in OMNIS data files directly. To store a list in a SQL table on a server, you can map it to a binary field of some kind.

Row

The *row* is a structured data type, like a list, that can hold multiple columns and types of data, but has one row only: it is essentially a list type with a single row. A row can have up to 400 columns. When you create and define a row variable, you set the type of each column. As with lists, the data type of each column in your row can be any one of the other data types including Character, Number, Date, Picture, List, and Row.

Object

Object classes let you define 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. A variable with *object* type is a variable based on an object class: the subtype of the variable is the name of an object class. For example, you can create an instance variable of Object type that contains the class and instance variables defined in an object class.

When you reference a variable based on an object class you create an instance of that object class. You can call its methods with the notation VarName.$MethodName(). For an object variable the initial value contains the parameters which are passed to $construct() for the class when the instance is constructed. The instance lasts for as long as the variable exists.

You can store object instances in a list. Each line of the list will have its own instance of the object class. You can store object variables, and hence their values, in an OMNIS data file or server database which can store binary values. If an object variable is stored in a data file the value of all its instance variables are stored in binary form. When the data is read back into OMNIS the instance is rebuilt with the same instance variable values.

Binary

The *binary* type can store structured data of unlimited length up to your maximum available memory. OMNIS does not know anything about the format and structure of the data in a binary column or field. In this type of column or field you could place, for example, desktop publishing files, MIDI system exclusive files, CAD files, and so on. You could store the definition of an OMNIS class in a binary field.

Binary data corresponds to binary large objects (BLOBs) on most database servers.

Item Reference

You can use a variable of type *Item reference* to store an alias or reference to an object in OMNIS or in your library. You assign the notation for the object to the item reference variable using the *Set reference* command. You can use an item reference variable in calculations or expressions which saves you having to quote the full path to the object. You can also use an item reference variable with the *Do* command to return a reference to the object or instance created by the command.

Field Reference

You can pass a reference to a field using the *field reference* data type, available for parameter variables only. A parameter variable with the field reference type must have a valid field in the calling method. Once the field reference parameter variable is set up, a reference to the parameter is the same as using the field whose name is passed.

Nulls and Empty Values

A variable or column of any data type can be NULL. This means the value is unknown or irrelevant, and that there is therefore no way to operate on the column value. A null value is distinguishable from an empty value, which represents empty or uninitialized data.

When defining a file class, you can specify that a field **Can Be Null** or **Cannot Be Null**. This controls the handling of rows written to OMNIS data files only and is irrelevant for client/server data, since it doesn’t prevent fields from getting null values in OMNIS calculations. Null data from a SQL database corresponds to null values in OMNIS fields and variables, and null values are sent to a server database as SQL nulls.

You can use the hash variable #NULL to represent null values in calculations. For example, to set a variable to null

```
Calculate LV_Variable as #NULL
```

The result of arithmetic, comparison, and logical operators on null data is always null. With string functions such as con() and jst(), however, OMNIS translates null to empty. The *isnull*() function returns kTrue if the value is null and kFalse if not.

When you use an OMNIS sort on columns or variables with nulls, OMNIS sorts the nulls first and separately from the empty values (or, for a descending sort, last). In a sorted report the nulls come first and do generate a break.

When exporting records in a text format, null values export as an unquoted string NULL, unless a particular format doesn't support nulls. In this case, OMNIS translates the null to empty. Occurrences of this unquoted string in an import file import as nulls.

Formatting Strings and Input Masks

You can further structure Character, National, Date, and Boolean data for display in window fields using formatting strings and input masks: see the *Window Programming* chapter in the *Omnis Programming* manual for details.

Variables

Variables can hold different types of data and are visible in different parts of your application depending on their data type and scope. For example, if you create a variable of list data type in a window class, the list variable and hence its data is visible within the window class and all its instances, but is not accessible elsewhere in your library.

Declaration and Scope

A variable may be global, accessible from all parts of your application, or it may have its scope restricted to certain areas so that it cannot be referred to from elsewhere. By declaring variables in the proper scopes, you limit the potential for arbitrary connections across your application and thus reduce the potential for error and the complexity of your application.

When two or more types of variable use the same variable name, a reference to that variable could be ambiguous. In a situation where more than one variable of the same name exists, OMNIS automatically uses the variable with the smallest scope. Therefore, it is possible, though not good practice, to have local, class, and task variables called "MYNAME". As OMNIS resolves ambiguity, a reference to MYNAME will refer to the local variable if it exists for the current method.

The following table lists the different kinds of variables and their scope. It also shows when they are initialized and destroyed.

Variable	When Initialized	When Destroyed	Scope
Parameter	on calling the method	returning to the calling method	the recipient method
Local	on running the method	on terminating a method	the method
Instance	on opening an instance	on closing the instance	a single instance of a class
Class	on opening a library	on clearing class variables or closing a library	the class, and all instances of the class
Task	on opening an instance of the task	on closing the task instance	the task, and all its classes and instances
Hash	on starting OMNIS	on quitting OMNIS	global

Apart from hash variables which are permanently built into OMNIS, you must create all variables with the appropriate type and scope in the objects in your library using the method editor. After you have declared them, variables that are in scope are listed in the Catalog. You can remove a variable using the Delete Variable option in the variable pane context menu. Declared variables are removed from memory when they are destroyed.

Parameter Variables

You can use a parameter variable to receive a value in a method, for example, a value that has been passed to the method using the *Do method* or *Do code method* command. You would normally do something with the value in the method and possibly return a new value. Parameter variables are visible within the called or recipient method only. They are initialized when the method is called, and cleared when the method returns to its caller.

Local Variables

Local variables are local to the method. You can refer to the variable within that method only. Local variables are initialized when the method begins execution, and are cleared automatically when the method terminates.

Instance Variables

Instance variables are visible to the instance only, that is, all methods and objects in the instance. You can define an instance variable only for classes that can be opened or instantiated: tasks, tables, windows, reports, menus, toolbars and objects. Note that you cannot declare instance variables in code classes. There is a set of the declared instance

variables for each instance of a class: these are initialized when the instance is constructed and cleared when the instance is destructed.

Class Variables

Class variables are visible within the class and all its instances. You can declare class variables for tasks, tables, windows, reports, menus, toolbars, and code classes. Any object or method in the class can refer to a class variable, and all instances of the class also have access to the class variable.

Class variables are not automatically cleared from memory. You can remove them from memory by closing the library containing the class, or using the *Clear class variables* command.

Task Variables

Task variables are visible within the task, all its design classes and instances. In practice, you can refer to a task variable from any method within any class or instance that belongs to the task. OMNIS initializes task variables when you open the task: for the startup task this is when the library opens. Note that you cannot declare a task variable for a class until you have set the **designtaskname** property for the class.

Hash Variables

OMNIS has a built-in set of global variables, called *hash variables* since they start with the symbol "#". You can view them in the Catalog.

– **#1-#60**, which are numeric variables

– **#L1-#L8**, which are list variables

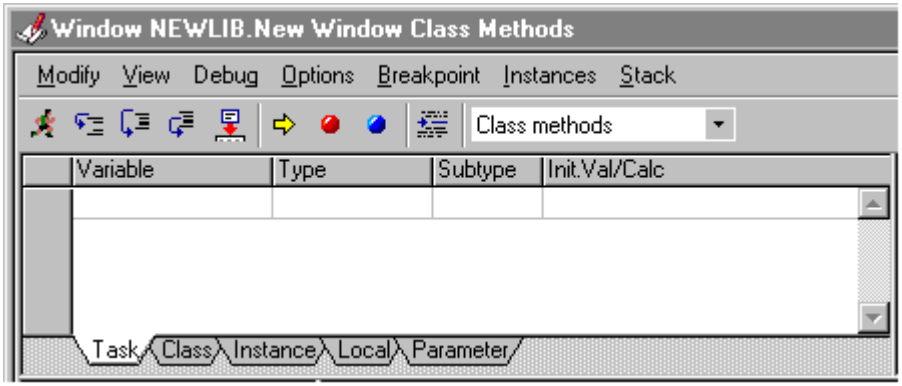
– **#S1-#S5,** which are string or character variables (up to 10,000,000 characters)

– **System**, which are miscellaneous values that OMNIS uses

Hash variables are global, unlike any other variables, so all libraries have access to them. The advantage of having global variables is that you can use these variables to pass data between libraries in an application. The disadvantage is that any data you place in hash variables remains there when you switch between libraries or combine libraries, with potentially unpredictable results.

Adding a Variable

You add variables to a class or object in the variable pane of the method editor. If the variable pane is not visible you can show it using the View>>Show Variable Panes menu option on the method editor menu bar. By default when you open a method editor for a class or object the Class tab in the variable pane is selected.



The tabs in the variable pane let you define task, class, instance, local and parameter variables; note that the local and parameter tabs only appear after you have added or selected a method in the method editor. You can add up to 400 variables of each type to the current object, including 400 local and parameter variables for each method in the current object. The name, type, subtype, and initial value of each variable is listed in the variable pane. You can size the columns in the variable pane by sizing the column headers.

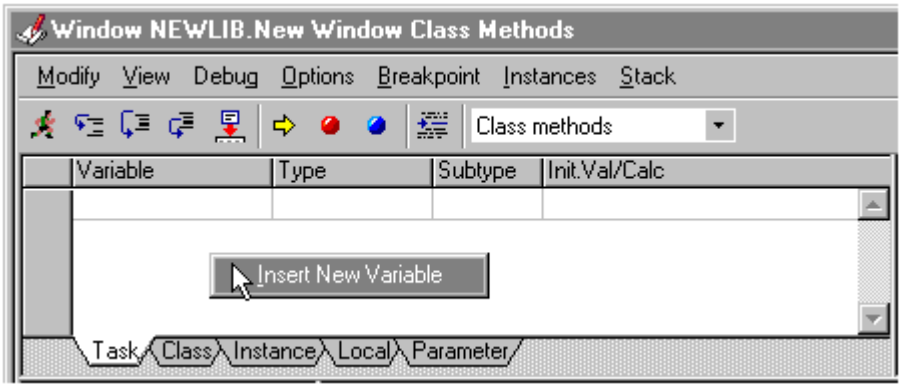
You cannot declare a task variable within a class until you have set the designtaskname property for the class: see the section below on *Adding Task Variables*.

To add a new variable

- Open the class in which you want to add the variable
- Right-click on the background of the class to open the class context menu
- Select the Methods option to open the method editor
- Choose the tab for the type of variable you require
- Click in the blank field under the Variable column header
- Enter the name of the variable

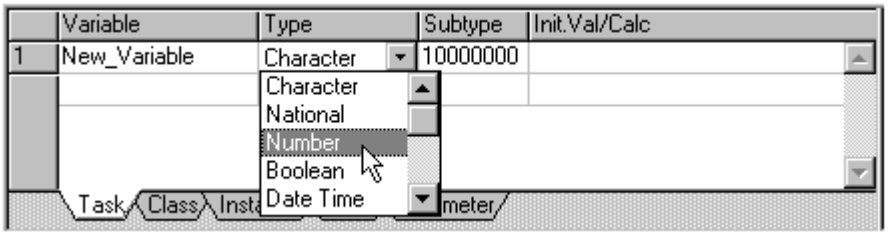
or

- Right-click in the variable pane to open the variable context menu
- Choose Insert New Variable and click in the variable name to edit it, or type over the new variable name if it is selected



Variable names can be up to 255 characters long, although you should keep them as short but descriptive as possible. When you name a variable you can prefix its name with one or more letters to indicate its scope. For example, parameters variables can begin with “p”, local variables “lv”, instance variable “iv”, and so on. This convention is not obligatory, but it may make your code more readable and help you identify variables throughout your library.

- Tab to the Type box and choose the type from the droplist using the mouse or arrow keys



or when the focus is in the Type box

- Type the first letter(s) of a data type to select it, for example, you can type “nu” to select the Number data type, or “b” for Boolean and “bi” for Binary type

For Number and Date Time variables

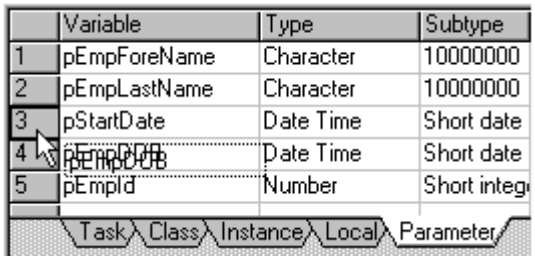
- Tab to the Subtype box and choose a subtype; again, you can type the first letter(s) of a subtype, for example, for Numbers you can type “L” to select the Long Integer subtype

You can enter an initial value or calculation for all types of variable. The initial value allowed for a variable depends on its type. See *Variable Values* below.

Adding Local and Parameter Variables

Local and parameter variables are inserted into the currently selected method. Therefore to insert these variables for a particular method, you need to select the method before inserting local and parameter variables.

Parameter variables receive values from a calling method in the order that they appear in the variable pane. You can reorder parameter variables by dragging them into position in the variable pane. Click and drag the fixed-left column or row number for a parameter variable in the list.



Normally you must declare all types of variable, including local variables, in the variable pane before you can use them in your code. However you can declare a special type of local variable in your code without first declaring it in the method editor. To declare such a variable, prefix the variable name with %% to create a string variable, or prefix the variable name with % for a numeric variable of Floating point type. You can type such variable names directly into your code in the method editor, at which time their names are added to the Local variables pane.

Adding Task Variables

To add a task variable for a class you *have* to set its **designtaskname** property. In most cases, the design task for a class is specified as the Startup_Task by default. You can change it using the Property Manager or the notation. The design task for a class is ignored at runtime.

To set up the design task for a class

- Click on the class in the Browser
- Display the Property Manager or bring it to the top, either from the Browser View menu or from the class context menu
- Click on the droplist in the **designtaskname** property to view the current tasks

The list of tasks will contain a Startup_Task, and any tasks you may have created.

- Select the design task by clicking on it

You will now be able to define task variables for this class.

Changing the Scope of Variables

You can change the scope of a variable at any time by dragging the variable from one variable pane to another. For example, you can change a class variable into an instance variable by dragging and dropping it onto the instance variable tab. Note you cannot change the scope of task variables.

To change the scope a variable

- Drag the variable from its current pane by clicking and dragging in the fixed-left column or row number for the variable
- Drop the variable on the appropriate tab in the variable pane

Variable Values

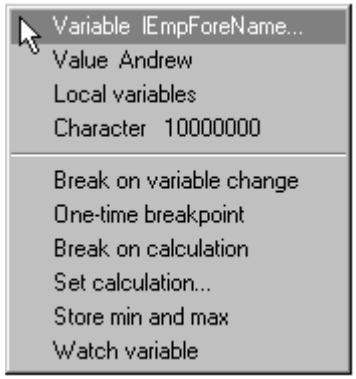
When you declare a variable in the variable pane of the method editor you can assign it an *initial value*. The first time a variable is referenced in a method, OMNIS assigns the specified initial value to the variable. You can set the initial value to be a number, string, calculation, some notation, or another variable name. In the latter case, when you first use the variable it gets the value in the other variable, regardless of the order of declaration.

For class variables only, the *Clear class variables* command clears the current values in all class variables and resets them to their initial values.

You can set the initial value of parameter variables, which in effect gives them a default value, but when and if a value is received in the method the initial value is overridden. For example, you may want to assign an initial value of zero to a parameter value to avoid it being null if a value is not received.

Variable Context Menu

You can lookup and edit the value of any variable or constant in OMNIS at any time using its context menu. You can Right-click on a variable wherever it appears in OMNIS to open its context menu and view its current value. The Variable context menu displays the variable name, its current value, which group of variables or class it belongs to, and its type and length. You can also perform various debugging functions from this menu as well.



If you select the first option in the Variable context menu, OMNIS opens a variable window containing the current contents of the variable which you can edit. Note that you cannot edit binary variables.

Variable Tips

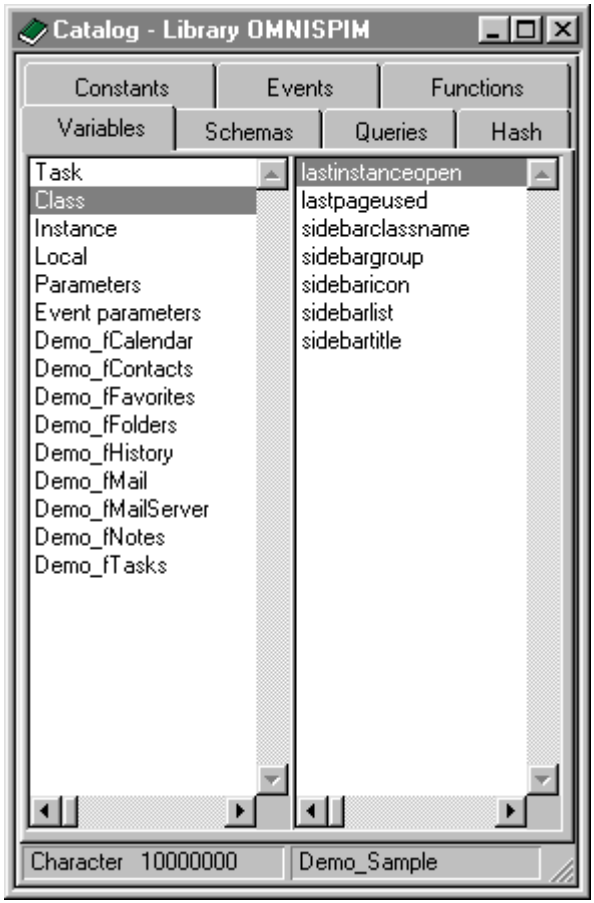
You can pass the mouse over a variable or constant and a *variable tip* will pop up displaying the variable's current value. Variable tips are available wherever variable names appear in OMNIS including the method editor and Catalog. However, they are not available if Help tips are enabled for the tool containing the variable. For some variable types, such as binary variables, the tip may say "not empty" which tells you the variable has a value, but it is too long to display.

Viewing Variables in the Catalog

You can view the variables in your library and the current class using the Catalog.

To view the variables in your library

- Open the Catalog (press F9/Cmnd-9) and click on the Variables pane



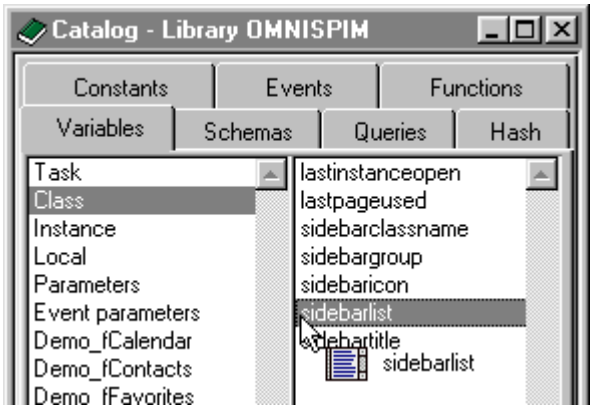
The Variables pane shows all the Task, Class, and Instance variables for the current class, plus all Local and Parameter variables for the currently selected method. Following the Event Parameters group, the Catalog also lists any file classes in your library. You can enter the name of any variable that appears in the Catalog into your code either by double-clicking on the name in the Catalog, or by dragging the variable name out of the Catalog into the method editor.

To select a variable from the Catalog

- Open a method editor for the class and click the cursor where you want to enter the variable name, in the *Do* command Returns field for example
- Open the Catalog and click on the Variables tab
- Click on the group in the left column according to the scope of variable you want
- Double-click on the name of the variable in the right column to enter it into the method editor

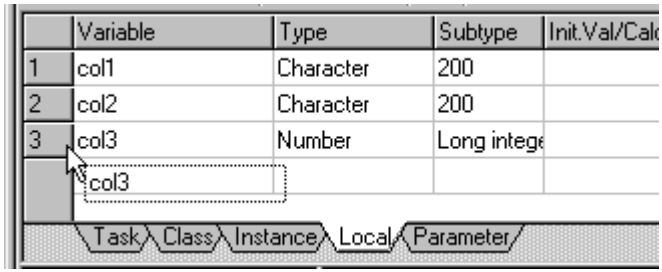
or

- Drag the variable from the Catalog and drop it onto the method editor



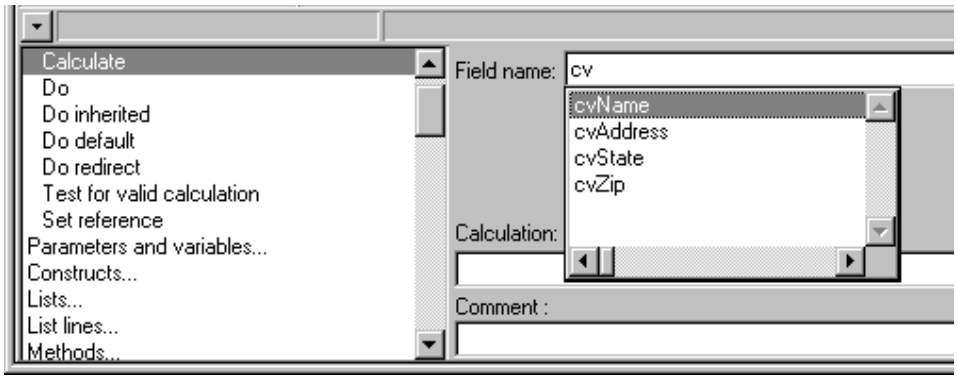
When you drag a variable from the Catalog, OMNIS shows you what type of variable it is and its name. Note that you can also drag variables from the Catalog and drop them onto window and report classes to create a data field for the variable.

You can also drag a variable from the variable pane in the method editor to any calculation or entry field in the command palette. To drag a variable name you need to click and drag the fixed-left column or row number in the variable list.



Auto Fill Variable Option

When you want to enter a variable in the method editor command palette and you can't remember its full name, you can type the first few characters of the variable, press tab, and choose the variable from the list that pops up. The list contains all the variables beginning with the characters you typed.



Methods

Most classes have their own default methods that you can execute using the *Do* command and the notation, but you can add your own methods to an object using the *method editor*. All types of class except schema, query, file, and search classes can contain methods. Methods you add to a class are called *Class Methods*. They control the behavior and functionality of the class, or the instance when the class is instantiated.

In addition to class methods, you can add methods to fields or objects in window, report, menu, and toolbar classes. These are called *Field Methods*, or in the case of menus they are called *Line Methods*, and for toolbars *Tool Methods*. Field methods are largely used for handling events generated by the objects in your application.

You can add up to 501 methods to each class, and to each field or object in a class. You can access the method editor in several different ways, including Right-clicking on the background of a class or field and selecting the Methods option from the context menu.

Viewing Class Methods

For window, menu, toolbar, or report classes

- Open the class in which you want to add a method
- Right-click on the background of the class to open the class context menu
- Select the Class Methods option to open the method editor

or you can

- Double-click on the background of the window, menu, toolbar, or report class

or for code, task, table, or object classes

- Double-click on the class in the Browser

Viewing Field Methods

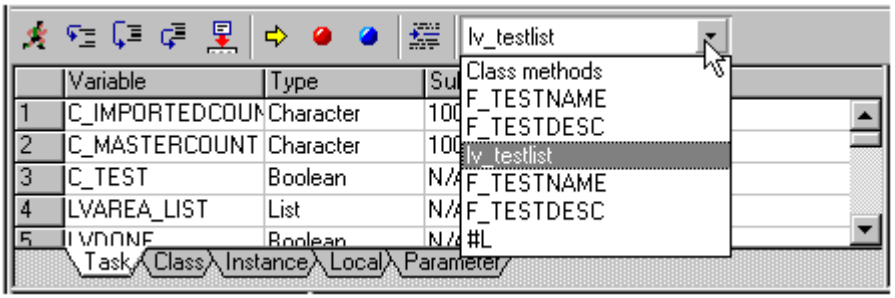
For window and report fields, menu lines, and toolbar controls

- Open the class in which you want to add a method
- Right-click on the field, menu line, or toolbar control to open a context menu
- Select the Field, Line, or Tool Methods option to open the method editor

or you can

- Double-click on the field, menu line, or toolbar control

There is a droplist on the method editor toolbar that lists all the field methods in the current class. You can select a field name in this list to display the methods for that field: you can also get back to the class methods from this droplist too.



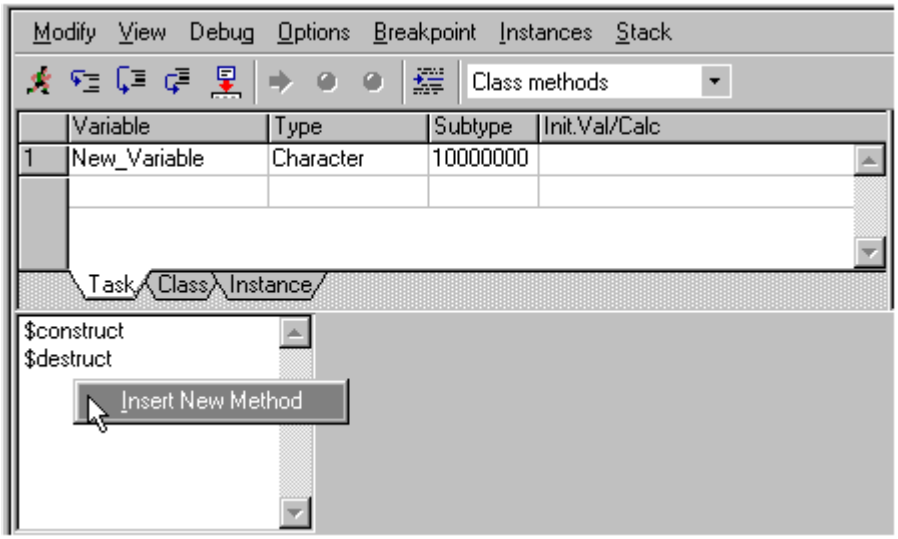
For menu classes the droplist lists the menu line methods, and for toolbars it lists the methods for each toolbar control, as well as the methods for the class.

Adding a Method

- Open the method editor for the class, field, menu line, or toolbar control, as above
- Click on the Insert new method button on the method editor toolbar; if the toolbar is not visible select View>>Toolbar>>Top from the method editor menubar

or you can

- Right-click on the left-hand side of the method editor, in the method list



- Select Insert New Method from the context menu

A new method appears with the default name selected.

- Edit the name of the method

Your own methods can have any name you choose. For example, you can call a method OpenCustomerWindow, PrintInvoice, DoInsert, and so on. The method name is the name you use to call the method from elsewhere in your library, so the name should describe what the method does. Method names should not include non-alphanumeric characters such as slashes, commas, and periods: this avoids possible confusion or errors when you call methods from outside the current library or when using the notation. Method names can begin with a "$" in which case they can be used in the notation: such methods are called *custom methods*. See the *Object Oriented Programming* in the *OMNIS Programming* manual for further details about custom methods.

Adding Code to a Method

Methods can contain one or more lines of code. You add the code for a method line on the right-hand side of the method editor, in the method definition pane. A single method can contain up to 240 separate lines of code. These lines of code can contain

– commands and functions

– OMNIS notation

– different programming constructs, and comments

OMNIS has over 500 commands that let you do almost anything in your application. A command can be a very simple instruction like the *Open window instance* command which opens the specified window, or a command can perform a complex operation like the *Do method* command which calls another method and returns a value. Constructs are specific programming structures such as *While*, *Repeat*, and *For* loops that combine a number of separate commands to control program flow or perform complex data manipulation.

OMNIS supports a wide range of functions that you can use in your methods. For example, you can manipulate dates and times, perform mathematical operations such as sin() and log(), and return a wide range of system specific information using sys().

Some of the OMNIS commands set a variable called the *flag* to true or false depending on the success of an operation. Other commands test the current value of the flag and branch accordingly. The *OMNIS Studio Reference* documents whether or not a command affects the flag.

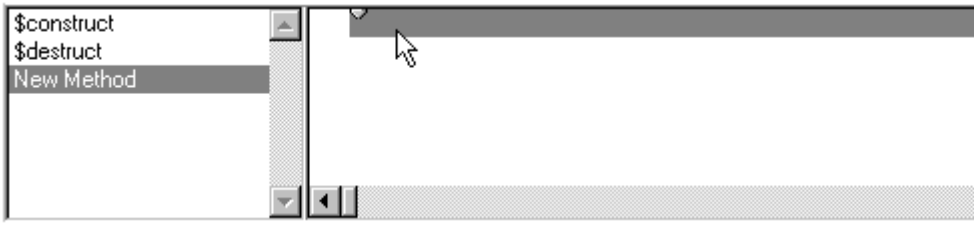
Comments can either be in-line comments, on the same line as the command, or a *Comment* command on a whole line. You can comment out command lines (commands are displayed as comments) to stop them executing, and uncomment lines of code as required. To make the code more readable on screen, you can use chromacoding which highlights different elements (constructs, general commands, comments and variables) in the method in different colors and styles.

The method editor lets you select multiple lines or entire methods and copy them between libraries or to and from a text editor. Also you can print methods to the current print destination.

You can add the code in your method from the keyboard, using point and click, or by pasting code in text format from the clipboard.

To enter a method from the keyboard

- Add a new class or field method as above, or select the required method name
- Press tab to enter the method definition area, or you can click to the right of the methods names list to highlight this area



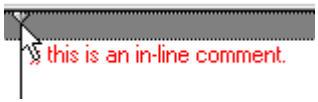
The top line is now the active area and has the focus.

- Type the first few characters of the command

The characters typed are a search string, for which OMNIS finds the first matching command; for example *c* finds *Calculate*, *en* finds *End If*. If you go wrong, you can press Backspace to clear the last character you entered. You can use an asterisk to represent intermediate characters, so *o*w* finds *Open window instance*. You can use the + and - keys to find the next or previous match for the string. A beep indicates there are no further matches. Typing *;* finds the *Comment* command.

Most commands require parameters or options. Parameters may be calculations or text, variable or constant names, or some notation. Some commands let you enter parameters from a list displayed in the command parameters box. Check boxes let you select options.

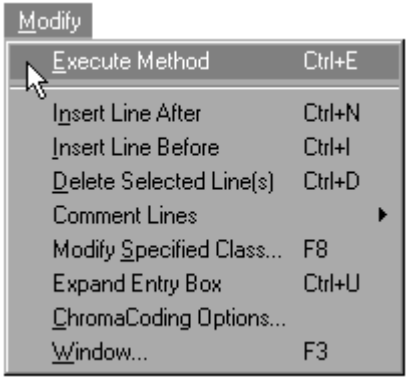
The Comment line is for in-line comments, indicated by ';;'. They normally start just after the end of the command, but you can line them up on the screen by dragging the indent marker across the screen.



The command line in the method shows options in parentheses and parameters in braces.

```
Yes/No message Warning (Icon,Sound bell) {Are you sure?}  ;; confirm
```

When writing a method you can use the commands in the Modify menu on method editor menubar, or their equivalent keys. The most useful is Ctrl/Cmnd-N for inserting a new command line *after* the current line.



- **Execute Method** (Ctrl/Cmnd-E)
  runs the currently selected method
- **Insert Line After** (Ctrl/Cmnd-N)
  inserts a new command line below the current one and tabs to the command list for the next command
- **Insert Line Before** (Ctrl/Cmnd-I)
  inserts a new command line above the current one
- **Delete Selected Line(s)** (Ctrl/Cmnd-D)
  deletes the selected line or lines
- **Comment Lines>>Comment Selected Lines** (Ctrl/Cmnd-;)
  comments out the selected method lines; **Uncomment Selected Lines** (Ctrl/Cmnd-')
  changes selected commented lines back into commands (if they are valid code)
- **Modify Specified Class** (F8/Cmnd-8)
  opens the class or method specified in the currently selected method line
- **Expand Entry Box** (Ctrl/Cmnd-U)
  opens a window for entering long command parameters such as SQL statements

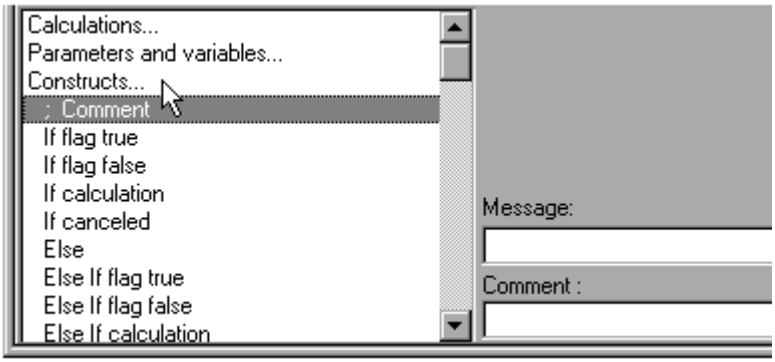
– **ChromaCoding Options**
sets the colors and style for command parameters: see the Customizing section at the end of this chapter

– **Modify This Class (F3/Cmnd-3)**
This option is applies to window, menu, toolbar, and report classes. When selected it opens the appropriate class design screen for the current set of methods shown in the method editor.

To enter a method using point and click

- Add a new method as above, and make sure the required method name is selected
- Double-click on the required group in the command list, indicated by an ellipsis, such as *Constructs...*

The list expands to show the commands in the group, and the focus jumps to the first command in the group.



- Scroll the command list and click on the command you want

Selecting a group and pressing Return also expands the group, and the up/down arrow keys choose commands. When you select a command, its parameters are shown to the right of the command list.

- You can tab to any of the parameter entry fields and enter a value or variable name

Entering a method from a text editor

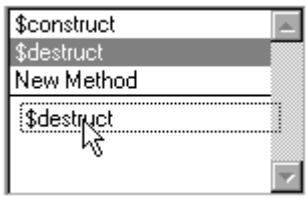
You can paste in command lines from a word processor, text editor, or the OMNIS help system or on-line manuals, but if the syntax is incorrect the method lines will be commented out. In particular, if the code contains variables that do not exist in the current method, some lines may be commented out. In this case, create each variable in the variable pane with the appropriate name, scope, and type, and try uncommenting the commented lines using Ctrl/Cmnd-‘ (apostrophe).

Editing a Method

You can edit a method whenever you like. Select the method you wish to edit by selecting it in the list of methods, and proceed as for adding a method. Note that you can insert lines into the code for a method using Ctrl/Cmnd-I., and you can use the Edit menu to Cut or Copy and Paste selected method lines.

Reordering and Renaming Methods

OMNIS sorts the methods in a class or field in alphabetical order and places any methods that begin with $ at the top of the method list. Furthermore, inherited methods are placed above any non-inherited methods.

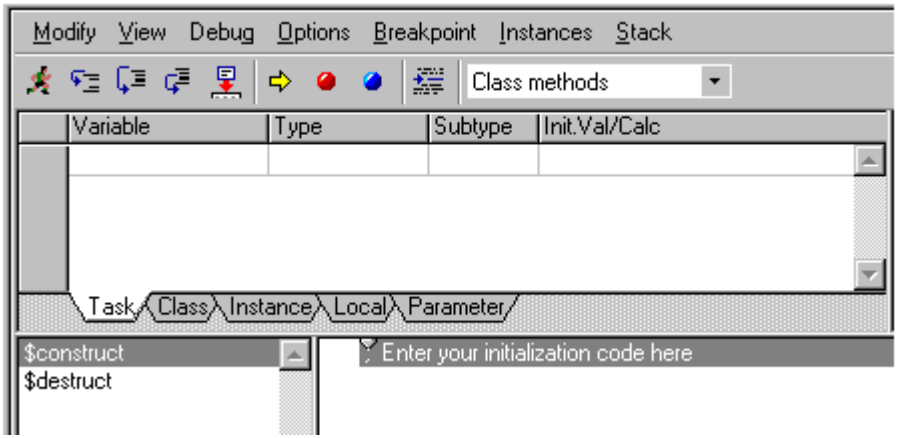


You can drag a method or a number of methods to reorder them in the method editor. After you have changed the ordering by dragging, OMNIS no longer sorts the methods automatically, retaining your ordering.

If you need to rename a method you should use the Find and Replace tool to rename it and all references to it. You can click on a method name in the method list and rename it, but all references to this method will be incorrect. When and if you rename a method, the method list is not resorted until you close and reopen the class.

Construct and Destruct Methods

When you open a class, an instance of the class is created. Most classes that you can open or instantiate contain two methods called $construct() and $destruct(), by default. This includes window, menu, toolbar, report, and table classes. If you open the class methods for one of these types of class you will see the $construct() and $destruct() methods in the method editor. Task and object classes do not have a $destruct() method.



The $construct() method controls the opening or construction of the instance, while the $destruct() method controls the closing or destruction of the instance. Just before a class is opened or instantiated its $construct() method is called. Therefore you can place in the $construct() method any code that you want to be executed as the class is opened. 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 will be executed as the instance is closed or destructed.

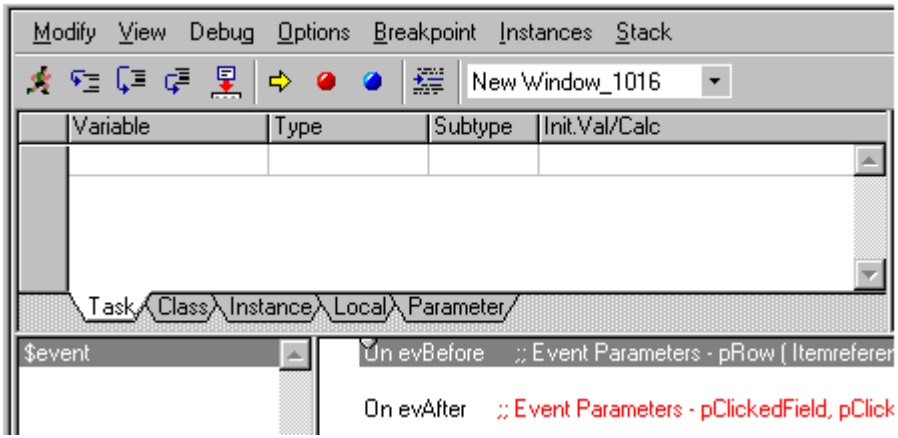
When you open a class, either using a command or the notation, you can send parameters to the $construct() method for the class. For example, if you open a window using the *Open window instance* command you can send *construct parameters* which are sent to the $construct() method in the window.

Note that you do not call the $construct() or $destruct methods using the *Do method* command to literally open or close an instance, rather these methods are called automatically when the class or instance is opened or closed.

Fields can also contain a $construct() method. These are called in the order of the fields on the window, and before the window $construct() method. You can use the $construct() method for a field to change the field as it is instantiated. You enter a $construct() method for a field in its Field Methods, not the Class Methods.

Event Methods

Window, report, menu, and toolbar classes can contain field methods. These primarily handle the events for the objects in the class: entry fields, list boxes, pushbuttons, menu lines, controls, droplists, and so on. Most window field types, menu lines and toolbar controls contain a method called $event(), by default, and all report fields contain a $print() method. If you open the field methods for a window, menu, or toolbar class you will see the $event() method, and if you open the field methods for a report you will see the $print() method. For example, if you open the field methods for a standard entry field you will see the following



Events are detected in the event handling methods using the *On* command plus the appropriate event message. Each type of event is reported as a unique event message. All event messages are prefixed with “ev” plus the name of the event. For example, when the user enters an entry field an evBefore event is generated, and when the user leaves a field an evAfter is generated. The default $event() method for an entry field contains two event handlers, *On evBefore* and *On evAfter*, which will intercept the evBefore and evAfter events. Any code that you enter after each of these event handlers will run when the event occurs and the appropriate event message is received.

Other types of window field will contain different event handlers. For example, a tab pane field contains an event handler to detect which tab is clicked on.

For further details about writing event handling methods and controlling events in your library, see the *Events and Messages* chapter in the *OMNIS Programming* manual.

Inherited Variables and Methods

A subclass inherits the variables, methods, and properties from its superclass. Specifically, a class inherits any custom properties and methods defined in its superclass, as well as its task, class and instance variables.

When you edit the methods for a subclass, you will see its inherited variables and methods in the method editor. These are shown in color and are listed at the top of the method and variable names list as appropriate. When you select an inherited method, the code for the method is not shown, and OMNIS will not let you enter code into an inherited method.

You can override an inherited variable or method by creating one with the same name as the inherited one, or you can override a variable or method using its context menu.

To override an inherited variable

- Right-click on the variable in the variable pane of the method editor
- Select the Override Variable option from the context menu

To override an inherited method

- Right-click on the method in the method editor
- Select the Override Method option from the context menu

Alternatively for methods only, if you create a method with the same name as an inherited one, the new one will override the inherited one: the inherited one will not appear when you next open the method editor for the subclass. The new method will be empty, but will contain the same parameters as the method in the superclass.

When you override a variable or method, all code in the subclass which refers to the variable or method in the superclass, now refers to the new variable or method with the same name. However you can still refer to a variable or method in the superclass using the $inherited property, that is, $inherited.varname will reference a variable in the superclass, and $inherited.methodname will reference a method in the superclass.

You can edit an inherited method, or you may want to open the method editor for a superclass to add a new variable in the superclass.

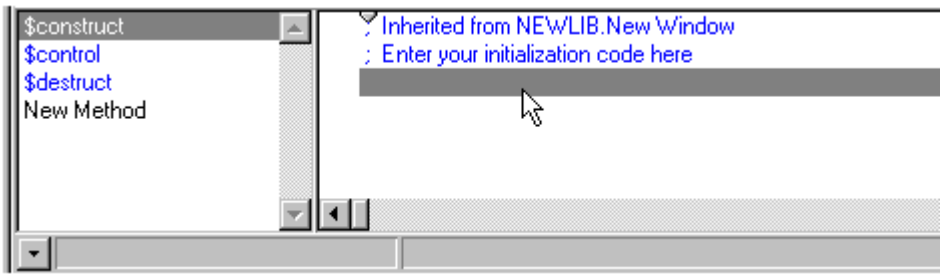
To open the methods for a superclass

Assuming you have opened a class that has a superclass

- Right-click on an inherited variable or method name
- Select the Superclass Methods option from the context menu

or, you can

- Double-click on the method definition area of an inherited method



OMNIS opens the method editor containing the methods and variables for the superclass: this will be the class at the top of the inheritance tree, not necessarily the class directly above the subclass you were editing originally.

Code Classes

The code class exists as a repository of methods to be called from other classes. This makes it easier to organize your application with reusable code so that the same method might be called from, say, a menu and a toolbar. You can execute any method contained in a code class from within the same code class using the *Do method* command, or you can call a method in a code class from any other class using the *Do code method* command.

You *cannot* open a code class, therefore code classes do not have instance variables, or $construct(), $destruct(), or $event() methods. Also you cannot create a subclass based on a code class.

When you call a method in a code class the current instance does not change, that is, the instance that contains the *Do code method* command remains the current instance. Therefore you can use $cinst in the code class to refer to the calling instance.

To create a new code class

- In the Browser choose Class>>New>>Code

or

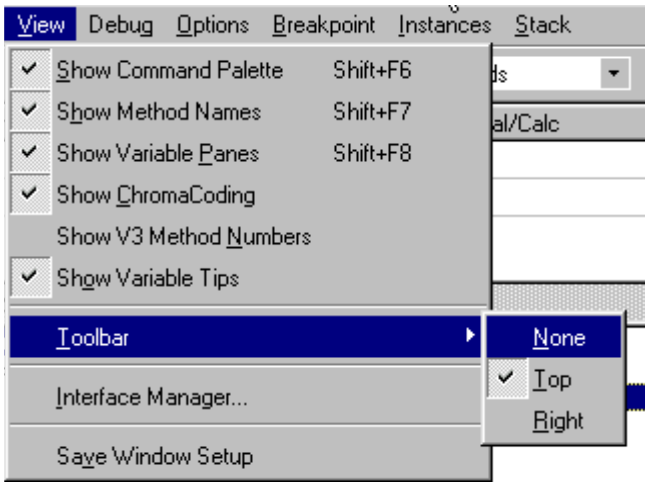
- Drag the template called “New Code” into the Browser from the Component Store
- Edit the name of the class

To enter code class methods

- Double-click on your code class, or select it and choose Class>>Modify from the Browser menu bar
- Right-click in the method list
- Choose Insert New Method and edit the method name
- Enter the method

Customizing the Method Editor

A number of options in the method editor let you change its appearance. Most of these are available via shortcut keys and the View Menu.



– **Show Command Palette** (Shift-F6/Shift-Cmnd-6)
shows or hides the command list and command parameters box

– **Show Method Names** (Shift-F7/Shift-Cmnd-7)
shows or hides the method list, enlarging the method definition pane

– **Show Variable Panes** (Shift-F8/Shift-Cmnd-8)
shows or hides the variable pane, changing the size of the method definition pane

– **Show ChromaCoding**
displays methods in the current chromacoding colors and styles

– **Show V3 Method Numbers**
displays the OMNIS 7 method numbers from converted libraries in parentheses after the method name

– **Toolbar**
displays the debugger toolbar at the top or right of the method editor, or hides it

– **Interface Manager**
opens the Interface Manager for the current class which lets you view the public methods for the class; you can add comments to each method to document the class

– **Save Window Setup**
saves the existing window configuration for the method editor. All changes that you make to the window, including changes to column widths in the variable pane, resizing

the height of the method pane, and so on are saved. After using this option, all further instances of the method editor are affected

You can change the colors used for chromacoding using the Modify>>ChromaCoding Options menu option in the method editor menubar. These options set the color and style (bold, italic) as follows.

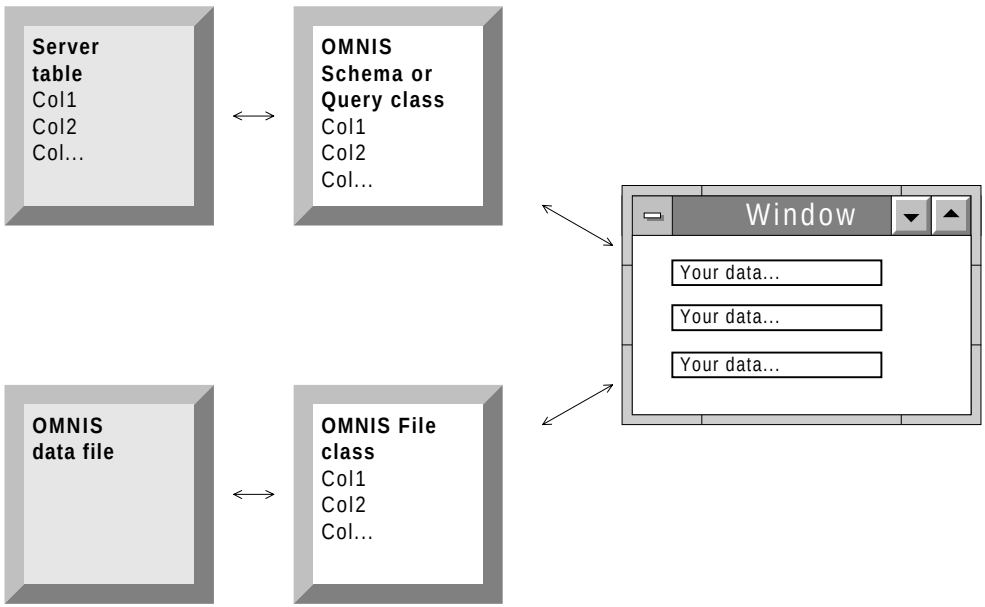
Properties	Affects
commentcolor commentstyle	comments
ctrlkeywordcolor ctrlkeywordstyle	constructs
keywordcolor keywordstyle	commands
stringcolor stringstyle	text strings
variablecolor variablestyle	variables

In addition to the above items the File menu option Print Selected Methods (Ctrl/Cmnd-P) prints the currently selected method or methods to the current print destination.

The method editor also has a comprehensive debugger. For more information about debugging your code, see the *Debugging Methods* chapter in the *OMNIS Programming* manual.

Chapter 5—Data Classes

Depending on the type of data you want to enter or retrieve in your client application you will need to define certain structures in your library to handle this data. These structures are stored in your library file as *data classes*. If you want to handle data from a server database you must create OMNIS *schema*, *query*, and *table classes* that map to the table and column structures in your server database. On the other hand, if you want to access data on your local disk, or use any non-client/server setup, you can store it in an *OMNIS data file*. In this case you need to design the structure for your data using OMNIS *file classes*. In addition, you can use an OMNIS *search class* to filter the data stored in an OMNIS data file.



This chapter introduces schema, query, table, file, and search classes, and shows you how to create them. All these classes are covered in greater detail in the *OMNIS Programming* manual.

All data classes in OMNIS use the same data types to represent data either on a server database or in an OMNIS data file. Before you create any data classes you should understand the different types of data you can represent in OMNIS. These are described in detail in the *Variables and Methods* chapter, but they are summarized below.

Data Types

You can use the following data types for schema columns and file class fields.

Character	standard character sets sorted by ASCII value
National	standard character sets sorted by national sort order
Number	multiple types for representing integers, fixed and floating point numbers
Boolean	single-byte values representing true or false values
Date Time	multiple types for representing simple dates and times, or composite date and times, between 1900 and 2099 to the nearest hundredth of a second
Sequence	proprietary data type for numbering OMNIS data file records
Picture	stores color graphics of unlimited size in a platform-specific format, or in the OMNIS shared picture format
List	structured data type that holds multiple columns and rows of data of any type
Row	structured data type that holds multiple columns of data in a single row
Object	uses the definition for its data from an object class
Binary	stores any type of data in binary form, including BLOBs
Item Reference	stores the full notation of an object in your library or OMNIS

Data Type Mapping

If you are building a client/server application, you need to create a set of schema classes that map directly to the tables or views on your server database. To do this successfully, you need to choose the data type for each column that best represents the type of data in your database server. See the chapters on client/server programming in the OMNIS Programming manual for more information.

Current Record Buffer

The Current Record Buffer, or CRB, is an area of RAM, that OMNIS uses to hold your current data. For example, if you are accessing a number of file classes or a SQL view, the CRB holds the current record or data for those files or view.

Schema Classes

A *schema class* is a type of data class that represents a table or view on your server database. A schema class contains the name of the server table or view on your server, and a list of column names and OMNIS data types that map directly to the columns in your server table or view. The OMNIS data types defined in your schema class should map to the equivalent server types, and the column names must conform to any conventions about case used by the server. For example, if the server column names are case sensitive, the column names in your schema class must be in the correct case. You can also provide a description for each column in the schema. Note that schema classes do not support null values.

Schema classes do not contain methods, and you cannot create instances of a schema class. You can however use a schema class as the definition for an OMNIS list using the $definefromsqlclass() method, which lets you process your server data using the SQL methods against your list. When you create a list based on a schema class a table instance is created which contains the default SQL methods. See the *OMNIS Programming* manual for further details about the SQL methods.

Having created a schema class, you can create a SQL form based on the schema to view and enter data into your server database. Creating SQL forms is described in the *Window Classes* chapter.

Creating Schema Classes

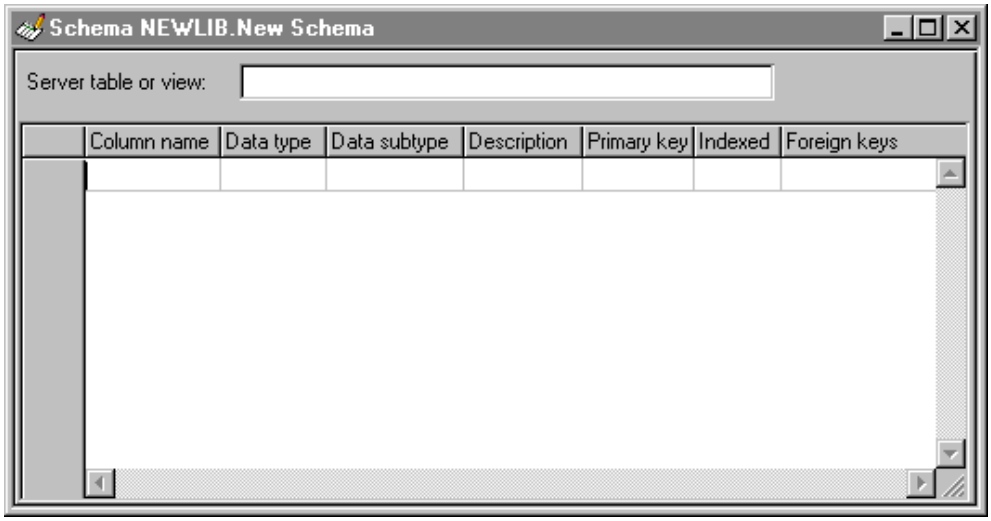
This section describes how you create a schema class from the Component Store or from the Browser.

To create a new schema class

- Drag the template called “New Schema” from the Component Store onto the Browser

or

- Select Class>>New>>Schema from the Browser menu bar
- Name the new schema class
- Double-click on the new schema to open the schema editor



The schema editor lets you enter the name of the server table or view and the column definitions You can move from column to column in the editor either using the Tab key, by clicking in the column, or with the keyboard Up and Down arrows.

To define a server column

- Type in a column name
- Select a type from the droplist of OMNIS data types
- Select a subtype if appropriate
- Add a description if required
- Select kTrue or kFalse to indicate whether or not this column is a Primary key; a Primary key column is a column that will be used in Where clauses
- Select kTrue or kFalse to indicate whether or not this column allows null values

Query Classes

A *query class* is a type of data class that lets you combine one or more schema classes or individual columns from one or more schemas, to give you an application view of your server database. A query class contains references to schema classes or individual schema columns.

Like schema classes, query classes do not contain methods, and you cannot create instances of a query class. You can however use a query class as the definition for an OMNIS list using the $definefromsqlclass() method, which lets you process your server data using the SQL methods against your list. When you create a list based on a query class a table instance is created which contains the default SQL methods. See the *OMNIS Programming* manual for further details about the SQL methods.

Having created a query class, you can create a SQL form based on the query to view and enter data into your server database. Creating SQL forms is described in the *Window Classes* chapter.

Creating Query Classes

To create a new query class

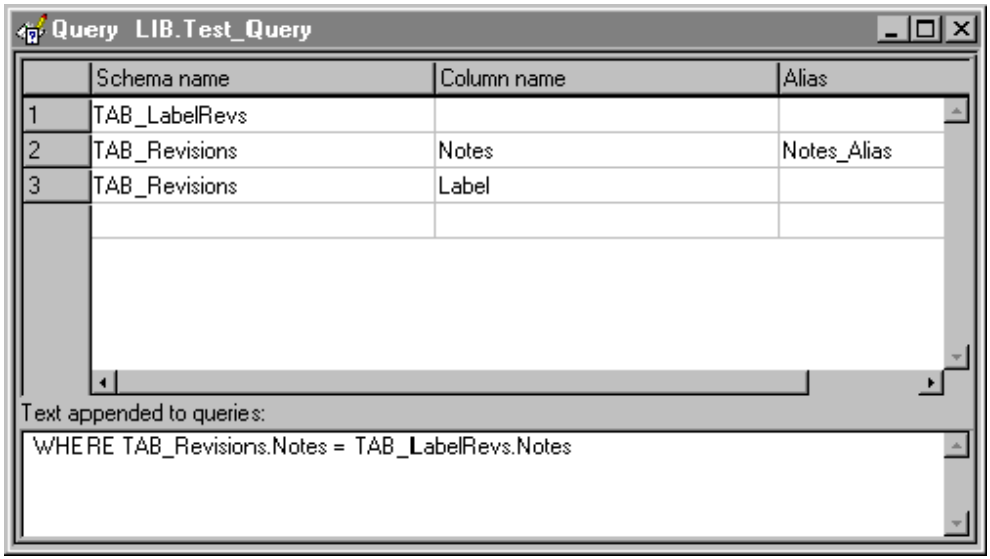
- Drag the template called “New Query” from the Component Store on to the standard IDE Browser

or

- Select Class>>New>>Query from the Browser menu bar

Then

- Edit the name of the new query class and press Return
- Double-click on the new query class, or Right-click on the query class and select Modify from its context menu
- Enter the names of the schema classes or schema columns



Note that the Catalog pops up when you open the query class editor, which lets you double-click on schema class or column names to enter them into the query editor. Alternatively, you can drag schema class or column names into the query editor. Furthermore, you can reorder columns by dragging and dropping in the fixed-left column of the query editor, and you can drag columns from one query class onto another. You can also drag a column from the schema editor to the query editor.

Table Classes

A *table class* provides the interface to the data modeled by a schema or query class, and exists primarily to allow you to override the default methods in the table instance. It contains the name of the schema or query class it uses, and your own custom methods that override or add to the default table instance methods. You can use a table class as the definition for an OMNIS list using the $definefromsqlclass() method, which lets you process your server data using the methods you added to the table class. See the *OMNIS Programming* manual for further details about the SQL methods.

You add methods to a table class in the method editor, and change its properties in the Property Manager.

Creating Table Classes

This section describes how you create a table class from the Component Store or from the Browser.

To create a new table class

- Drag the template called “New Table” from the Component Store onto the Browser

or

- Select Class>>New>>Table from the Browser menu bar
- Name the new table class

To associate the table class with a schema or query class

- Right-click on the table class and select Properties from its context menu

or you can

- Click on the table class in the Browser to select it
- Select the View>>Properties menu item from the Browser menu bar
- In the Property Manager, click in the **sqlclassname** property
- Type a schema or query class name, or select a class name from the droplist

To add methods to a table class

- Select the table class in the Browser and select the Class>>Methods menu item

or you can

- Right-click on the table class and select Methods from its context menu

or you can

- Double-click on the table class in the Browser

The method editor opens for the table class.

Creating SQL Classes Automatically

You can create SQL classes that map to your server tables or views automatically by dragging a server table from the SQL Object Browser onto your library in the IDE Browser.

For more details about logging on to a server database using the SQL Object Browser, and creating SQL classes automatically, please refer to the *Accessing your database* chapter.

File Classes

A *file class* defines the template for the types and lengths of the data to be stored in a record in an OMNIS data file.

Creating File Classes

This section describes how you create a file class from the Component Store or from the Browser.

To create a new file class

- Drag the template called “New File” from the Component Store onto the Browser

or

- Select Class>>New>>File from the Browser menu bar
- Name the New file class

File class names can be up to 255 characters long, and of mixed case.

- Double-click on the new file to open its editor

To modify a file class

- Select the file class in the Browser and select the Class>>Modify menu item

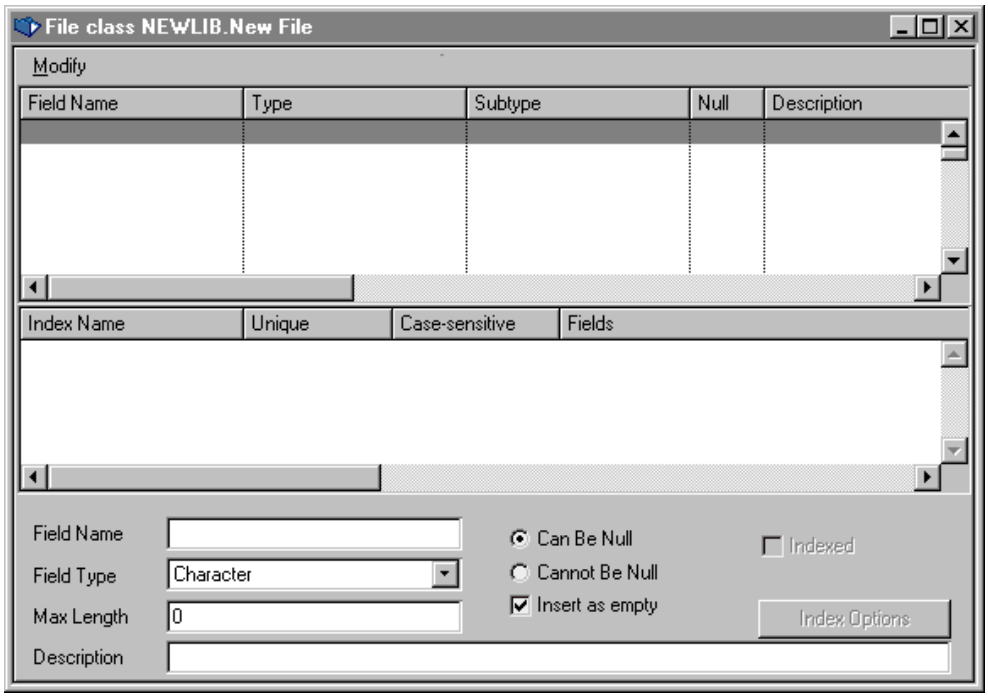
or you can

- Right-click on the file class and select Modify from its context menu

or you can

- Double-click on the file class in the Browser

The file class editor opens.



In the file class editor you define the fields for the class. The top part of editor lists the fields in the file class, the middle section lists the indexes for the file class, and you define the fields for the file class in the bottom section.

For new file classes the first line in the field list is selected, by default, and the cursor is in the Field Name box. Data types for fields are described in the *Variables and Methods* chapter.

To create a field

- In the Field Name box, enter the name of the field
- Select a data type from the Field Type droplist

for Character, National, and Binary type fields

- Enter the maximum length of the field in the Max Length box

for Number, Date Time, and Object type fields

- Select a subtype for the field in the droplist that opens under the Field Type droplist
- Add a short description for the field, if required
- Select the appropriate radio button depending on whether or not the field can be Null

If the field cannot be Null, OMNIS prevents a record being written to the data file if this field does not have a value.

- Check the Insert as Empty checkbox if you wish OMNIS to assign an empty value automatically to the field when a record is inserted
- Check the Indexed checkbox if you want the field to be indexed
- Click on the Index Options button to edit the index
- When you have edited an index, click on the Field Options button to get back to the field list (or you can click in the field list)
- Click on the next line of the field list, or press Ctrl/Cmnd-N, and enter the next field

When you close the file editor, OMNIS saves the field definitions for the file automatically.

Search Classes

You use *search classes* with your OMNIS data to select particular types of data for printing, or to restrict the number of records to be processed in an OMNIS method. For example, you can use a search class to select all the clients in a certain area, or you can select all employees with a salary above a certain amount.

Creating Search Classes

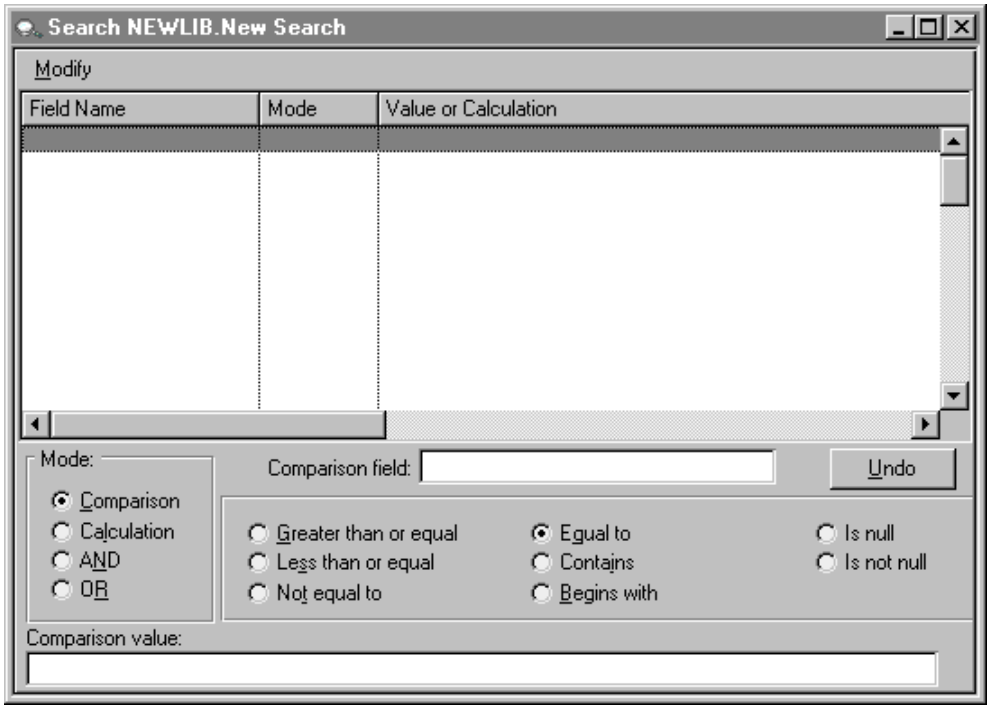
This section describes how you create a search class from the Component Store or from the Browser.

To create a new search class

- Drag the template called “New Search” from the Component Store onto the Browser

or

- Select Class>>New>>Search from the Browser menu bar
- Name the new search class, and double-click on the search class to edit it



A comparison line consists of three elements: a comparison field, a comparison type or mode such as equal to, and a comparison value. A line of the search class can be a comparison, a calculation, or a logical AND or OR. You enter the comparison fields, operators, and values at the bottom of the search editor. You can add up to 50 different comparison lines to each search class.

A search class takes the specified field in each record in your OMNIS data file and compares it with the value entered in the comparison value box. If a record passes the test or all the tests in the search class, it is selected for display or printing in a report, and so on. If a record fails any of the tests in the search class, it is rejected and the next record in your data file is tested.

To enter a comparison

- Click in or tab to the Comparison field box
- Enter a field name, or select one from the Catalog (press F9/Cmnd-9, locate the field in the file class, and double-click on its name in the Catalog)
- Select a search type or mode from Comparison, Calculation, AND, or OR
- Click on a comparison type from Greater than or equal, Less than or equal, and so on
- Click in or tab to the Comparison value box and enter text or calculation for the comparison

The characters you type into the comparison value box are read as a text string. You can use square brackets to force OMNIS to evaluate the contents of a field, for example, [FIELD2].

- Press Enter when you have finished the comparison line

Chapter 6—Window Classes

A client application is, from the user's point of view, a collection of windows or forms in which they enter or view their data. You define the appearance and behavior of each window in your application as a separate *window class*.

This chapter describes how you create windows or forms using the templates and wizards in the Component Store, how you add fields and other objects to a window, and how you modify these objects using the Property Manager.

Each window class can contain a number of fields, controls, and graphical objects which you can create from the Component Store. You can put an unlimited number of fields on a window, and you can open any number of windows, except under 16-bit Windows you are limited to a maximum of 50 open windows due to an operating system restriction.

The Component Store contains a number of window templates and wizards for you to use as a basis for your window classes. You can however add your own templates to the Component Store and change the default ones, and how to do this is described in detail in the *Library Tools* chapter.

Creating Windows using Wizards

This section describes how you can create a window class automatically, using a wizard from the Component Store. A window you create in this way contains a number of fields that map directly to schema, query, or file class fields in your library, which lets you view data on your server or OMNIS database. Before you can use window wizards you must create the schema, query, or file classes necessary for SQL or OMNIS data access; this is described in the *Data Classes* chapter. The following wizards are available

SQL Form Wizard
creates a form or grid based on a schema or query class; each separate field or grid column on the new window maps to a schema column

OMNIS Form Wizard
creates a form or grid based on a file class; each separate field or grid column on the new window class maps to a file class field

The SQL and OMNIS Form wizards let you select either a form or grid window to display your data. Other templates in the Component Store let you create basic forms or grids only, including

SQLColumns
standard form based on a schema class

SQLGrid
non-enterable grid based on a schema class

SQLSmartlist
enterable grid based on a schema class with Commit and Revert All buttons

SQLParChild
form that uses a parent-child or one-to-many join

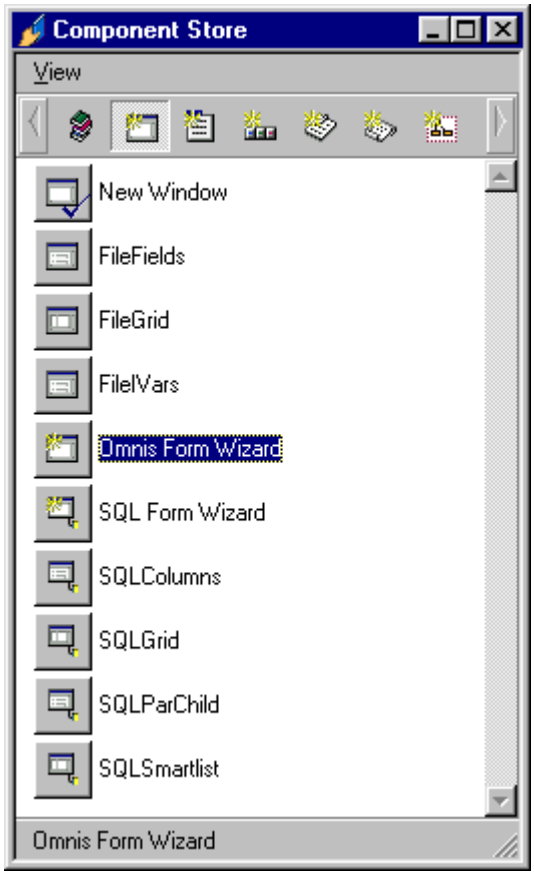
FileFields
form with one window field per file class field, plus standard OMNIS database controls such as Next, Find, Previous

FileGrid
grid window with one column per file class field

FileIVars
form with one window field per instance variable mapped to each file class field, plus standard OMNIS database controls; this window lets you display records in separate window instances

To create a window class using a wizard

- Open your library in the Browser, and display the classes using the View>>Down One Level menu option on the Browser menu bar
- Open the Component Store or bring it to the top using F3/Cmnd-3
- Click on the Window Classes button in the Component Store toolbar to show all the window templates and wizards



- Drag a SQL or OMNIS Form Wizard, or other window template from the Component Store onto your library in the Browser
- Name the new window class and press Return, or click in the Browser

The Window Wizard dialog lets you choose between a simple form or grid view, and is displayed for the SQL and OMNIS forms only; for some of the basic templates this dialog does not appear. The SQL Form wizard lets you choose a window style from the following

– **Field per column** from a schema or query class
standard form that displays your server data and contains one window field per SQL column; allows selection of individual columns, based on an OMNIS schema or query class that maps to your server table columns

– **Display grid** from a schema or query class
non-enterable grid that displays your server data; allows selection of individual columns, based on an OMNIS schema or query class

– **Enterable grid** from a schema or query class
enterable grid with Commit and Revert All buttons to view and edit your server data; allows selection of individual columns, based on an OMNIS schema or query class

– **Parent/child window** from two schema or query classes
form that uses a parent-child or one-to-many join; prompts you for a parent and child table, and parent key and child key; allows selection of individual columns within the tables, based on OMNIS schema or query classes

The OMNIS Form wizard lets you choose from a similar set of window styles, but based on file class fields for displaying your OMNIS data, as follows

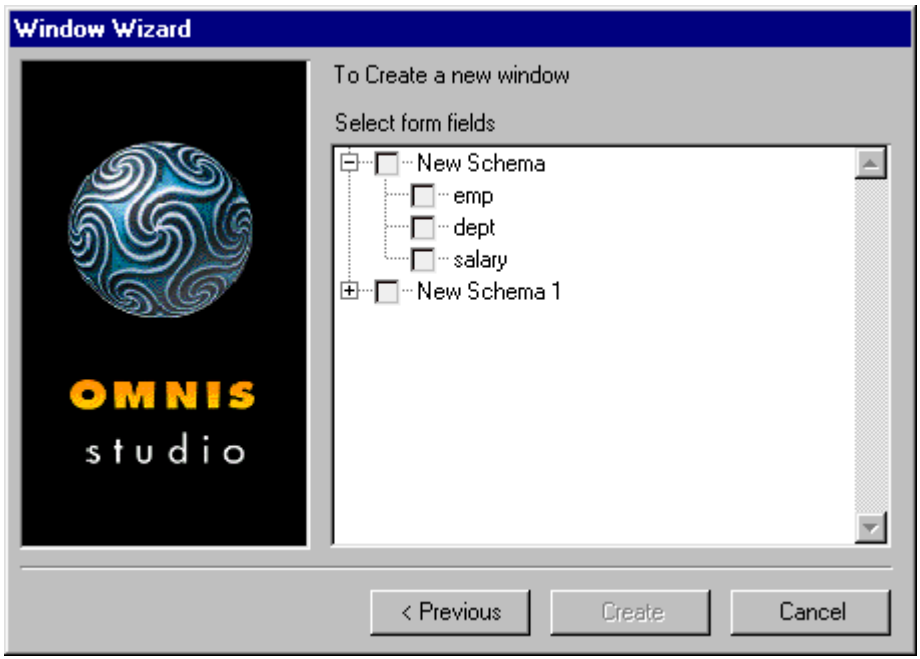
– **One field per file field**
  form with one window field per file class field, plus standard OMNIS database controls such as Next, Find, Previous

– **One instance variable per file field**
  form with one window field per instance variable mapped to each file class field, plus standard OMNIS database controls; this window lets you display records in separate instances of the window

– **Display grid based on file class**
  non-enterable grid with one column per file class field

If you click Cancel at any time, the wizard is halted and the new class is removed from your library. To continue

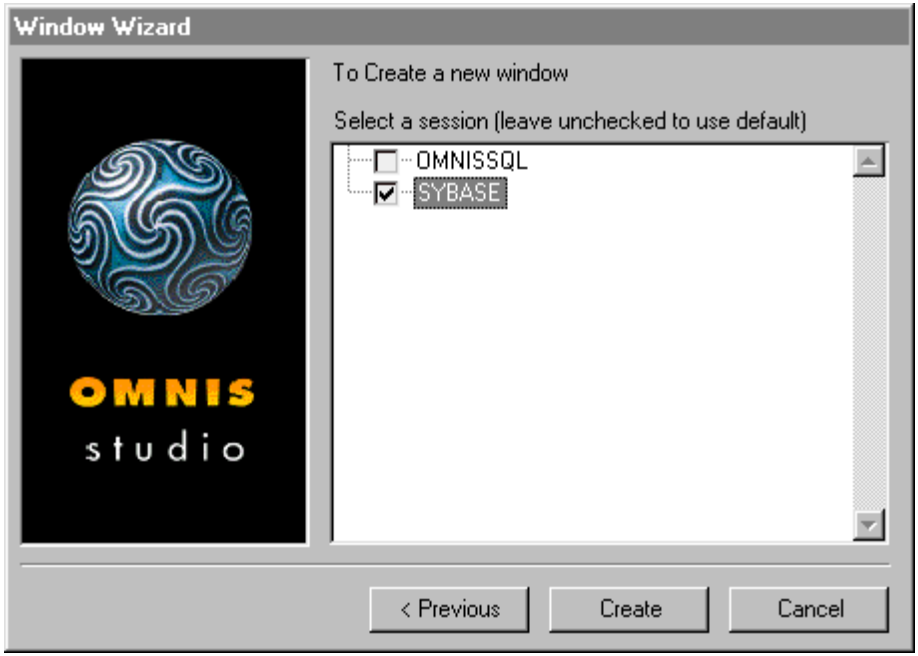
- Choose a window style, and click on Next



The Window Wizard now displays all the available SQL or file classes in your library and lets you select which table columns or file fields you want to include on your form. You can expand a particular table or file to include or exclude individual columns or fields, but you cannot include different columns or fields from different classes (although a query class

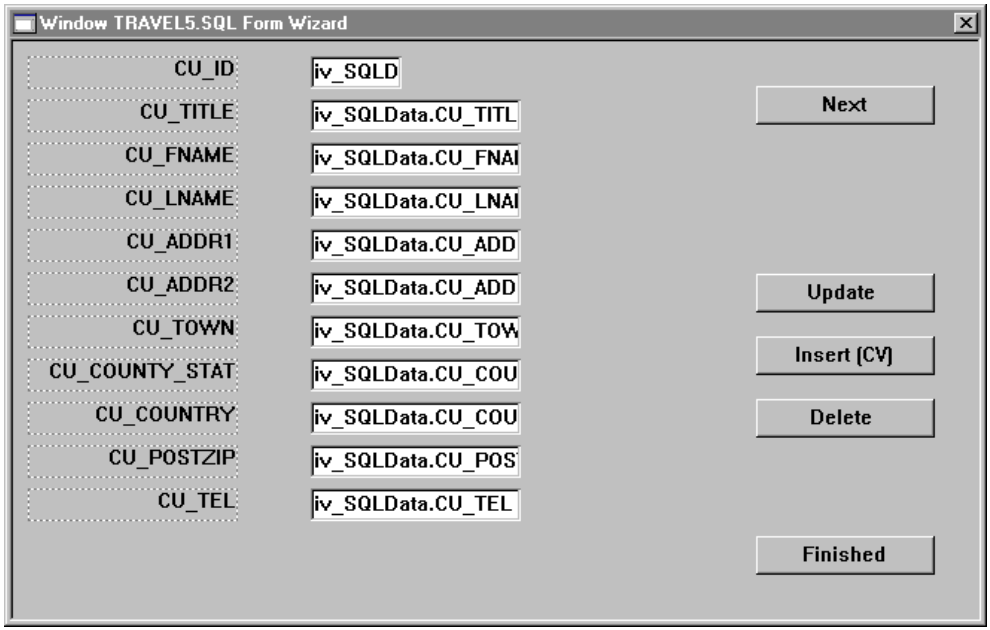
may include columns from more than one schema). If you check a SQL or file class name without expanding it, *all* the columns or fields from that class are included on your form.

- Choose a schema, query, or file class to base the form on, or select individual server columns or file fields in a class, and click on Next



For SQL forms only, the wizard prompts you to choose a session for the form, or you can leave the session names unchecked to use the default session.

- Finally, click on Create to create the form



When you finish in the Window Wizard the new window class is opened ready for you to modify or open to view your data. To modify your window you need to edit its properties. You can also add new window objects from the Component Store, and you can add methods to the window class or the objects on the window. These are all described in this and subsequent chapters.

To open or test your window

- Assuming the design window is on top, press Ctrl/Cmnd-T to open your window

or you can

- Right-click on the background of your window class, and select Open Window from the context menu

or from the Browser you can

- Right-click on the new window class, and select Open Window from its context menu

While you create and modify any window class you can use the Ctrl/Cmnd-T shortcut to toggle between the design and open window to see how it looks and behaves. Assuming you are logged on to a server or OMNIS data file, you can view and edit your data in the new window.

Default Window Template or Wizard

The *default window template* or wizard is the one used when you create a new class using the Class>>New>>Window menu option in the Browser, and it's also the class shown under the Default Classes button in the Component Store toolbar. The default window template appears under the Window Classes button in the Component Store, shown by a check mark on the template icon. However, you can change the default template for window classes.

To change the default window template or wizard

- In the Component Store toolbar, click on the Window Classes button
- Right-click on the class or wizard you want to make the default
- Select the Make Default option from the context menu

For example, you can make the SQL Form Wizard the default template and from thereon, when you use Class>>New>>Window in the Browser, the SQL Form wizard will be called by default. When you click on the Default Classes button in the Component Store, the template or wizard you set as the default is shown. The remainder of this chapter assumes that the class called "New Window" is the default class.

Creating a New Window

This section describes how you can create a window class from the Component Store or from the Browser using the New Window template. Unlike windows you create using the wizards, new windows created in this way contain no fields, methods, or any other window components, but this lets you create your window entirely from scratch.

To create a new window

- Open your library in the Browser
- Display the classes in your library using the View>>Down One Level menu option on the Browser menu bar
- Drag the template called “New Window” from the Component Store onto the Browser

or

- Select Class>>New>>Window from the Browser menu bar

The name of the new class is highlighted in the Browser, to continue

- Name the new window
- Double-click on the window class to modify it

When you create a new window from the Component Store or Browser it is completely blank, that is, it does not contain any data entry fields or controls, and has only a simple frame and title bar. Having created your window, you can change its style and appearance and start to add fields from the Component Store.

Window Types

This section describes the different types of window available in OMNIS and tells you how to change the style of your window using the Property Manager. The next section describes window properties and how you modify them to change the behavior of your window.

You can create many different types of window: you can change their size, you can add scroll bars, you can hide or show the title bar, you can choose a different frame style or a window can have no frame at all. Also you can change the background color and pattern of each window in your library. All these visual features are properties of the window. You can change these and many other aspects of the window appearance and behavior using the Property Manager. When you open a window class to modify it the Property Manager should open automatically showing the properties of the current window.

To view the properties of a window

- Right-click on your window name in the Browser
- Select the Properties option in the context menu

or

- Select your window name in the Browser and press F6/Cmnd-6

or, if you open a window to edit it and for some reason the Property Manager does not pop up, you can

- Click on the background of the window design screen

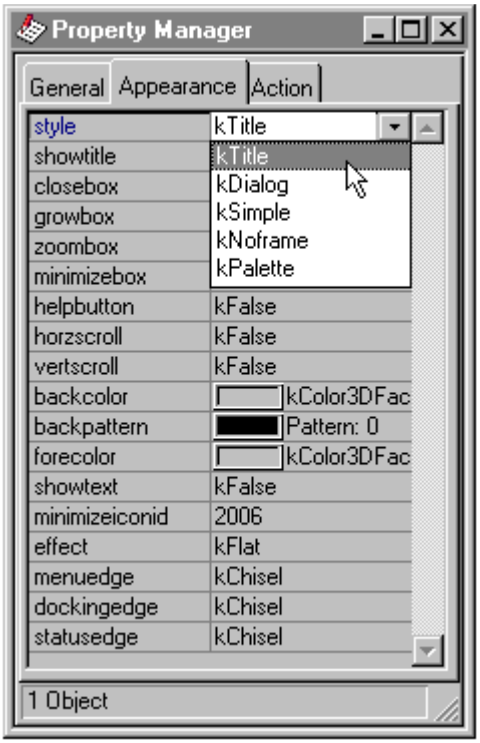
or

- Right-click on the background of the window design screen
- Select the Properties option in the context menu

The Property Manager shows the general, appearance, and action properties of the current window class.

To change the style of a window

- View the properties for your window class, as above
- Click on the Appearance tab in the Property Manager
- Click on the **style** property and select a style from the dropdown list



The Appearance properties of a window class

The other appearance properties control the window frame and its controls, that is, whether or not your window has scroll bars, a close box, zoom or grow box, as well as the color and pattern of your window background, and so on. Note that you cannot set some of these properties for some types of window.

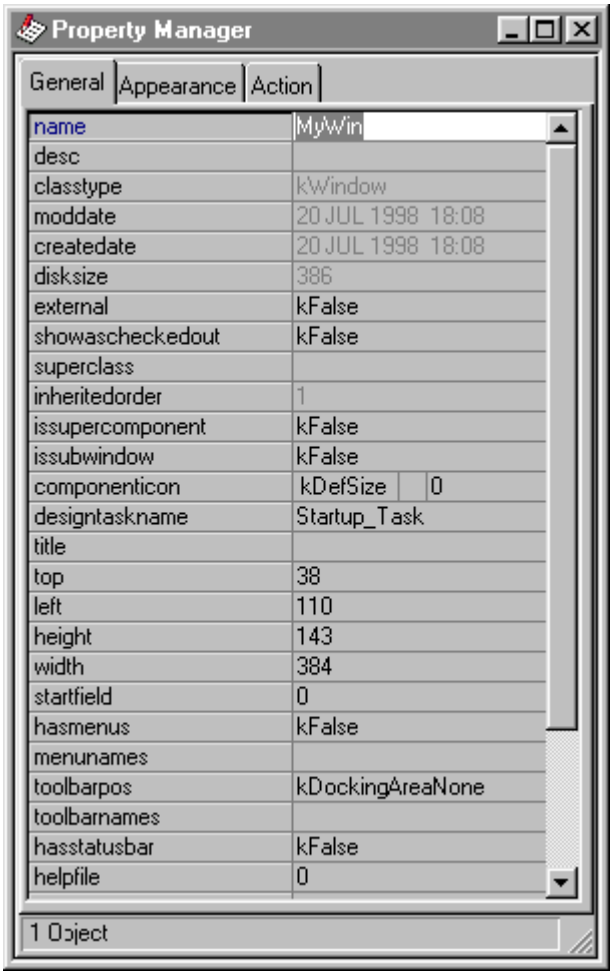
You can further modify the 3d effect of the window background for all types of window using the **effect** property. For example, you can make a window frame flat, inset, embossed, beveled, chiseled, or shadowed.

The Appearance tab contains the **edgefloat** property. When the size of the area available to window instances changes, open window instances and window classes float according to the value of their edgefloat property. The default value of this property is kEFnone.

Window Properties

This section describes the various window properties and how you modify these using the Property manager to change the behavior of your window. You can modify any window you create from the Browser or Component Store, including those you have created using a window wizard.

The General properties for a window determine the many visual aspects of the window including its title, size, and whether or not it has a menubar, toolbar, or status bar. Also you can set up a design grid in your window to help you place objects accurately. You may need to scroll or resize the Property Manager to view all the window properties.



The General properties of a window class

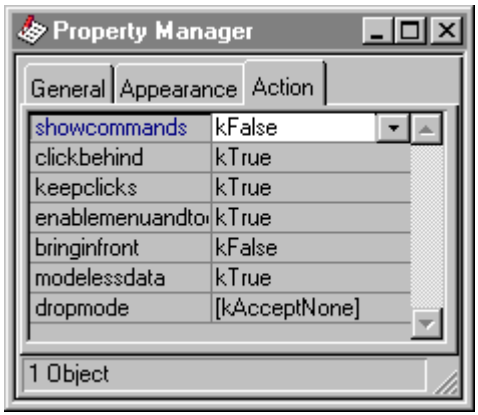
For details about the general class properties such as **name, desc, classtype** and setting the **designtaskname** property, see the *Libraries and Classes* chapter. More advanced features such as inheritance, adding help, or making your window a template or component are described in the *OMNIS Programming* manual. Many of the other window properties are self-explanatory, and in general you simply need to set the property to hide, show, or enable a particular window component. They are summarized as follows.

title	the window title
top	position of the top edge of the window in pixels
left	position of the left edge of the window in pixels
height	height of the window in pixels
width	width of the window in pixels
startfield	first enterable field on the window
hasmenus	enables the window menu bar
menunames	comma separated list of menus on the window menu bar
toolbarpos	position of docking area in the window
toolbarnames	comma separated list of toolbars for the window
hasstatusbar	if true the window has a status bar
helpfile	name and path of help file for the window
contextmenu	name of the context menu for the window
showgrid	if true the design grid is shown
aligntogrid	if true objects align to the design grid
sizetogrid	if true objects are sized to the design grid
horzgrid	horizontal size for the design grid in pixels
vertgrid	vertical size for the design grid in pixels

To create simple, easy-to-use data entry windows you can accept many of the default settings for the majority of window properties. The **top** and **left** properties position the window relative to the OMNIS application window. The properties **hasstatusbar**, **hasmenus** and **toolbarpos** control whether or not the window has its own status bar, menu bar, and toolbar respectively. A window toolbar can be at the top, bottom, left or right of the window. You can enter toolbar names as a comma-separated list in the **toolbarname** property, or drag them onto the window toolbar from the Browser. Similarly, you can drag menus onto the window menubar from the Browser. You can set the border style of the window menu bar, toolbar, or status bar by setting the **menuedge, dockingedge**, or **statusedge** properties under the Appearance tab. Double-clicking on the toolbar or on a menu title opens the appropriate editor. To remove a menu or toolbar, simply drag it off the window.

The Action properties for a window class need further explanation. These properties control the behavior of the window when you use the window to enter data. Several of the

properties let you make a window *modal*, that is, when such a window is opened it disables the main menu bar, it does not allow any clicks behind it, and it always appears on top of all other types of window, regardless of their properties or behavior. This means the user has to do something in your window, such as enter a value, before it will close or give them access to other parts of the application.



showcommands shows the Commands menu in the main menu bar when this window is opened or brought to the top: this menu contains standard OMNIS database commands such as Find, Next, Previous, etc.

clickbehind if true allows clicks behind this window, that is, it lets the user bring another window to the front of this one by clicking on it

keepclicks determines whether or not controls on this window receive clicks when it is brought to the front

enablemenuandtoolbars enables or disables the main menu bar, so you should use this with caution: when set to false, this property adds to the modal behavior of the window

bringinfront opens this window on top of all other windows, including palettes: again, this property adds to the modal behavior of the window

modelessdata if true the window is modeless instead of an enter data mode window: see below

dropmode determines the type of data the window accepts using drag and drop: this can be all types, or specific types including edit, list, or picture fields

Modeless Enter Data vs Enter Data Mode

In applications that use data files as opposed to the client/server interface, OMNIS needs to ensure that two users cannot edit the same record. It does this with *enter data mode*, a style of window behavior that forces methods to activate controls on the window to allow data entry. OMNIS provides standard database controls (Insert, Find, Previous, etc) in this mode of data entry and also installs the Commands menu with these commands when a user opens the window.

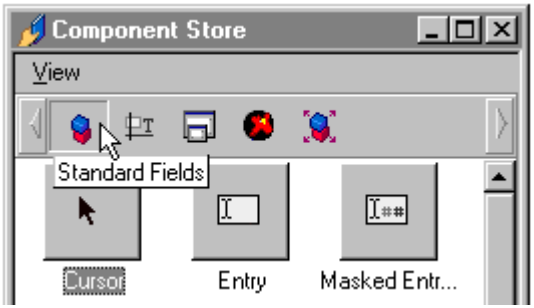
Client/Server applications, however, have transaction management built into the server and do not require this extensive, client-side record locking. When prototyping for client/server applications, you should instead choose *modeless enter data*, an alternative style of window behavior that leaves all fields enterable at all times, subject to the properties of the field. You need to code the SQL to query, insert, update, or delete rows from the server as methods behind the appropriate window buttons and controls.

Window Fields and Properties

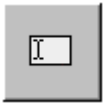
This section describes the different types of window field that are built into OMNIS, and tells you how to create them from the Component Store and the Catalog. It also describes the properties of window fields, and how you modify these to change the field's appearance and behavior. Graphical objects such as lines and ovals are described under the *Background Objects* section. You can also place external components on a window and these are described later in this chapter.

The single-most important property of a window field is its **dataname**, that is, the name of the variable, table column, or file class field the window field uses to display or insert its data. Almost all types of window field have a **dataname** property which you must set in the Property Manager when you create a field. The **dataname** for most types of grid or list field is the name of your list variable used to display data in the object.

When a window class is the top window, the Component Store contains window fields and other components; if the Component Store does not display fields try clicking on your window. The Component Store toolbar includes different groups of fields and components, including background objects, subwindow fields, and external components. The Standard Fields button is selected by default.

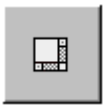


Under the Standard Fields button in the Component Store there are many kinds of OMNIS field, each with different characteristics and data handling capabilities.

Field type	Description
	**Cursor** deselects all buttons in the Component Store and returns the cursor to the pointer tool
	**Entry Field** a field for entering data or a read-only field for display only
	**Masked Entry Field** a standard entry field, but contains a formatting mask to control what the user can enter
	**Multi-line Entry Field** a multi-line field for entering data or read-only field for display only
	**Pushbutton** a control field that runs a method or command when you click on it
	**Button Area** an invisible control field that runs a method or command when you click on it
	**Radio Button** a toggle field for one out of several mutually exclusive choices

Field type	Description
	**Check Box** a toggle field for on-off, yes-no, 1 or 0 boolean choices
	**Picture** a screen area into which you can paste or display graphic images
	**Shape Field** a graphical object that has certain properties of a field including events
	**Subwindow** a field that contains another window class
	**Popup Menu** a field that pops up a menu when you click on it
	**Popup List** a list that pops up when you click on it and from which the user can make a choice
	**Dropdown list** a list that drops down when you click on it and from which the user can make a choice
	**Combo Box** a droplist that also lets you type in a value directly into the field
	**List box** a multi-line list field with scroll bars
	**Headed List Box** a list field with button style column headers and adjustable column widths.

Field type	Description
	**String Grid** a grid field for displaying character data
	**Data grid** a grid field for displaying character or numeric data
	**Complex Grid** a multi-line field with columns and rows
	**Check List** a list in which each line can be selected using a checkbox icon
	**Icon Array** a list field that lets you present a series of choices each represented by a large or small icon
	**Tree List** a hierarchical list with expandable and collapsable nodes
	**Tab Strip** a multi-tabbed control for switching the context of the objects on your window
	**Tab Pane** a multi-pane field with tabs that lets you add fields and other objects to each pane
	**Paged Pane** a multi-pane field that lets you add fields and other objects to each pane
	**Group Box** a field container with border and title text

Field type	Description
	**Scroll Box** a scrollable container field for other fields and objects
	**Modify Report Field** a field that displays a report class; lets users change certain aspects of the report
	**Screen Report Field** a field that displays a printed report in your window

You can place a window field on your window from the Component Store and set its **dataname** property in the Property Manager, or you can drag a variable or field from the Catalog to create a field automatically.

To create a window field from the Component Store

- Open your window in design mode
- Drag the required field type from the Component Store onto your window

or, to draw a field of a particular size

- Select the required field type in the Component Store by clicking on its icon
- Click and drag on your window to define the size of field you want

or, to place a field on your window automatically

- Double-click on the appropriate icon in the Component Store

When you double-click on an icon in the Component Store a field of that type will appear in the center of your window. You can repeat this as many times as you want to place multiple copies of the same type of object.

When you select a field in the Component Store, that button remains selected until you place the field. If you want to change your mind and not place the field, click on the Cursor control in the Component Store, and the previously selected tool will be deselected.

To create a window field from the Catalog

- Open your window in design mode
- Open the Catalog and locate your variable, column name, or file class field under the Variables tab
- Drag the required variable onto your window

When you place fields by dragging variables from the Catalog, OMNIS creates a field of the correct type and sets its **dataname** property to the name of the variable, column, or field. For example, if you drag a character or number variable onto your window OMNIS creates a standard entry field, if you drag a list or row variable OMNIS creates a list box field, and so on. You cannot drag some types of variable, such as item references and binary variables.

There are a number of restrictions that apply to dragging variables from the Catalog to create fields.

– You can drag fields of type date, number, sequence, character, boolean and picture only

– You cannot drag local and parameter variables

– You can only drop class and instance variables on to the class to which they belong

– You can only drop task variables on to classes belonging to the same design task

Examining and Changing Field Properties

When you place a field on a window, using any of the above methods, the Property Manager will open showing the properties of the field, but if for some reason the Property Manager does not come to the top you can open it in a number of ways.

To view the properties of a field

- Open your window in design mode
- Click inside the field and the Property Manager opens automatically

or

- Right-click on the field
- Select the Properties option in the context menu

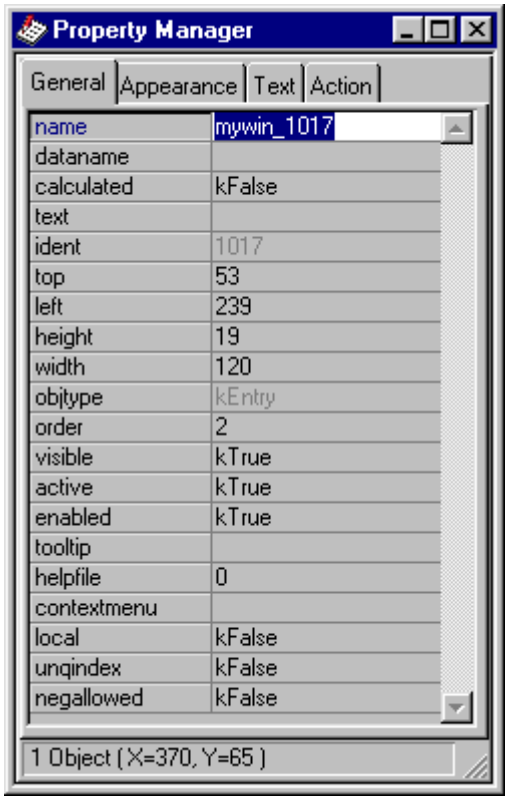
or to bring the Property Manager to the top

- Select the field, or any other object, and press F6/Cmnd-6

Entry Fields

An *entry field* is a type of window field into which you can insert data or view existing data. In OMNIS there are several types of entry fields available: standard *entry field*, *masked entry field*, and *multi-line entry field*. You can modify an entry field to create a *display* or *local* field that you can use to display data.

This section describes the properties of entry fields. Additional properties for such field types as tab panes, pushbuttons, and picture fields, are discussed separately.



The General properties of an entry field created from the Component Store

For entry fields the Property Manager shows General, Text, Appearance, and Action properties. Note that once you have placed a field on your window you cannot change its **objtype**. To create a field of another type you have to create a new field from the Component Store.

General Properties

Most types of field have these general properties and many of them are self-explanatory. These properties control the overall behavior and appearance of the field. All fields and window objects have a **name** which can be any name you choose to identify the object. There is no restriction on the name of an object, but you should try to use unique names within the same class to avoid possible confusion or errors when you refer to the object. All data fields have a **dataname** property, and all fields have a particular size and position, set up using the **left, top, width**, and **height** properties. Here is a complete list of general properties for an entry field and a short explanation for each.

name	the name of the field; this can literally be anything you want, although it has to be unique in the context of its window
dataname	the name of the variable, column, or file class field the window field uses to display or insert its data
calculated	for display fields only, computes the field value from the text or calculation in the text property
text	the calculation for calculated fields, also the text for a check box or radio button
ident	unique numeric identifier of the field: all objects are numbered consecutively on the window, including background objects
top	position of the top edge of the field relative to the window border in pixels
left	position of the left-hand edge of the field relative to the window border in pixels
height	height of the field in pixels. Set it to -1 to set the field height to its default value
width	width of the field in pixels: you can change this and the height property in the Property Manager to size the field exactly
objtype	the type of window field: once you create a field you cannot change its objtype
order	the tab order, that is, a number that controls the tabbing order of fields: you can renumber fields to change the tab order
visible	if true the field is visible: otherwise you can hide the field at runtime by setting this property to false
active	if true the field is active: if false, the user cannot enter the field, tab to it, or do anything else to it; this lets you completely disable the field except for mouse events
enabled	if true you can enter this field and change its data and so on: if false the field becomes a read-only display field

tooltip	the text to show in the tooltip for the field
helpfile	the name of the file containing the help topic for the field
contextmenu	the name of the menu class that pops up when the user Right-clicks on the field
local	if true the field recalculates and redraws when the previous, non-local field changes
unqindex	if true the field is checked for a unique index value (for indexed fields in an OMNIS data file only); when you try to leave the field, OMNIS checks that the value does not already exist in another record
negallowed	if true you can enter negative values

Most types of field have a **dataname** property. When you create a field that you want to contain or display data, you must set this property. This establishes a link between the field on your window and the variable, column, or file class field that the window field represents. You can set the dataname property for a field in the Property Manager, or when you create a window field from the Catalog, as described above, the dataname is set for you automatically.

For entry fields, the **dataname** is a simple variable or field name as defined in the class. For grid and list fields the dataname is the name of the list variable that contains the list data. Popup menu fields do not have a dataname: you specify the name of the menu class in the **menuname** property. Pushbuttons do not have a dataname: you use buttons as controls hence they do not contain data. Other types of field such as group boxes and shape fields are for screen layout only and do not have a dataname, since they cannot contain data. The **unqindex** property is for window fields that access OMNIS data only, not server data.

Masked entry fields have the additional general properties

formatstring	data entry formatting string for the field
formatmode	type of formatting string for the field, either Character, Integer, Number, Date
inputmask	input mask for the field

The **formatstring** property stores a set of characters or symbols that formats the data in a masked entry field for display, regardless of how the data is stored. The **inputmask** property contains a string that formats data as the user enters it into a field. When a user enters data into a field controlled by an input mask, OMNIS rejects any characters that do not conform to the format you've specified in the mask.

To enter a format string for a masked entry field, you need to specify the type of data represented in the field by setting its **formatmode** property: you can set this to Character, Number, Date, or Boolean. You can enter a format string manually or use one from the dropdown list in the **formatstring** property. The default formats in this dropdown are stored

in a system table described in the *Window Programming* chapter in the *OMNIS Programming* manual.

Text Properties

You can change the font style, size, and color of a window field's text by setting its text properties. You can set the **fontstyle** property to plain, bold, italic, underline, and so on. The **textcolor** property can be one of the color constants or an RGB value.

fieldstyle	the field style for the object: OMNIS has some default styles for standard entry fields, pushbuttons, and lists, but you can define your own or change the default ones, described in the *Window Programming* chapter in the *OMNIS Programming* manual.
font	font type for the field
fontsize	font size for the field
fontstyle	font style for the field
textcolor	text color of the field
align	alignment or justification of the field
subwindowstyle	applies to fields in subwindows; if true the field uses the text styles defined for the subwindow field, otherwise if false it uses its own text styles defined in the subwindow class

Appearance Properties

You can change the overall appearance of a field by setting its appearance properties. Different fields have different properties, but the most common are listed below. The color properties take a system color constant or an RGB value which you can choose from a color picker in the Property Manager. To set the colors for a pattern, you need to set the foreground and background color.

forecolor	foreground color of the field
backcolor	background color of the field
bordercolor	border color of the field; only applies to certain effects
backpattern	background pattern of the field
edgefloat	floating edge property for the field: you can make the top, left, right, or bottom edge of a field floating, or combinations of these: also you can size a field to fit into certain positions in the container field, such as toolbars, status bar, menu bar, or the available client area
dragborder	whether you can drag the field's borders or not; only applies to fields with a kEFposn... edgefloat property set
horzscroll	for single-line entry fields allows the data in the field to scroll horizontally; for multi-line entry fields, enables a horizontal scroll bar

	for the field
vertscroll	for multi-line entry fields, enables a vertical scroll bar for the field
uppercase	converts all text to upper case characters
zeroempty	displays zero values as blanks
shownulls	displays "NULL" for undefined values
linestyle	line style for the border of the field; only applies to certain effects
effect	the border style for the field; for example inset, beveled, chiseled or shadowed

Note that the appearance of grid and list fields is discussed in the *Lists and Grids* chapter.

Action Properties

The action properties let you set up the drag and drop modes of a field. This feature lets you select data, drag it to a different part of your application, and drop it onto another field or window.

dragmode	turns on drag and drop for the field: you can drag the *data* in a field, or the field itself, or a duplicate of the field
dragrange	drag range or scope of what you drag from the field, a constant
dragiconid	id of the icon for the field when it is dragged
dropmode	specifies the type of data this field accepts
autotablen	the number of characters the user can type before OMNIS tabs to the next field automatically
autofind	the field performs an automatic find (for indexed fields in an OMNIS data file only); when you leave the field, OMNIS looks for a matching record in the indexed data file field you specify

Password Entry Fields

Single line entry fields have a property **passwordchar** which specifies the character to be displayed for every character entered in the field. When the property is set, the data in the field cannot exceed 255 characters, and while the focus is on the field the Cut and Copy items on the Edit menu are disabled.

Local Fields

A *local field* is a field that depends on the value of the prior field in the tab order. OMNIS redraws these fields immediately after redrawing or changing the prior field. You usually use local fields to display data changed as a result of an entry in a preceding field. OMNIS will not automatically execute the field procedure on recalculation. You can have more than one local field running in sequence after a non-local field.

Using a local field after a list box, you can set up a spreadsheet-like edit bar for a selected list line. When you select a line in a list, the local field changes to display that line; you can then edit the line and put the updated line back into the list.

A calculated display field following a field you specify as part of the text or calculation should have the **local** property to ensure up-to-date display of the display field value.

Display and Inactive Fields

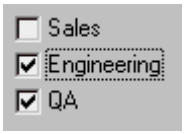
A *display field* is a type of window field that you use to display data only, that is, the user cannot enter data into a display field. To change a standard entry field into a display field you change its **enabled** property to false; to display data in the field you set its **calculated** property to true and enter the data in its **text** property. The user can't tab to a disabled display field or click in it and enter data, but a display field still accepts mouse events, such as mouse leave events. To make a field completely inactive you need to change its **active** property to false, regardless of its **enabled** setting. Such an inactive field does not receive mouse events and you cannot enter data into it.

Entry and Display Field Calculations

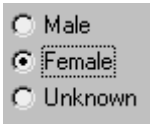
You can specify a validation expression for the data in the field. You can use input masks to force the user to input data in certain basic formats, but more complex logical constraints require an expression. To make a calculated field you must set its **calculated** property to true. You can enter an OMNIS expression into its **text** property. When the user leaves the field, OMNIS validates the data using the expression. If the expression evaluates to false, OMNIS beeps and returns the cursor to the field. For display fields, the **text** property lets you enter a character or numeric value or OMNIS expression that is displayed in the field.

Check Boxes and Radio Buttons

Check boxes represent boolean data, that is, they can display On or Off choices, Yes or No, and 1 or 0 values. Radio buttons present a number of mutually exclusive buttons that can be either on or off: selecting one of the radio buttons deselects all other buttons in that group. You must number radio buttons consecutively in the tabbing order.



Check boxes



Radio buttons

The field you associate with the check box should be a number or Boolean field. The field you associate with radio buttons should be numeric. Checking a check box sets the value of the field to one; unchecking it off sets the value to zero. Clicking a radio button sets the value of the field to zero for the first button, one for the second button, two for the third button, and so on.

You enter the text to display to the right of the check box or radio button in the **text** property for the object. OMNIS calls the field method for check boxes and radio buttons when you click on the field.

Pushbuttons and Button Areas

Pushbuttons are control fields that activate either user-defined methods or standard OMNIS database commands such as Find, Next, and Previous. When you click on a user-defined button, OMNIS sends the appropriate message to the button and runs its $event() field method. Button areas behave in exactly the same way except that they are invisible on an open window (shown with a dotted or gray line in design mode). Button areas let you place an invisible and clickable control on top of a graphic, or behind the whole window.

You set the text for a pushbutton in the **text** property. Under Windows, you can use the "&" character before a letter to specify a key to use with Alt to push the button from the keyboard instead of with a mouse. For example, if you specify the text "&Cancel Tour", you can use the Alt-C key combination to activate the button.

Pushbuttons have some additional Appearance properties.

nogray	if true the button does not gray when inactive
noflash	if true the button area does not flash when clicked (button areas only)
buttonstyle	the drawing style of the button
iconid	id of the icon used for picture buttons

And some extra Action properties.

buttonmode	mode or type of pushbutton or button area; buttons are user-defined by default which means you can add your own method
actedata	if true the button is active during enter data; note the button will not work if this is set to false, in particular on modeless enter data windows
actnomethod	if true the button is active when no methods are running; note the button will not work if this is set to false
inactnorec	if true the pushbutton is inactive when there is no current record (applies to OMNIS data files only)

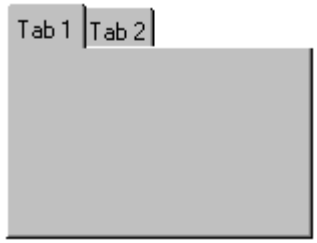
When you create a pushbutton from the Component Store its **buttonmode** is kBMuser or user-defined by default. This means you can enter your own method behind the button which will run when the user clicks on the button. Other button modes run standard methods including the OMNIS database commands such as Insert, Find, Next, and OK.

Pushbutton Picker Palettes

You can place color picker, line style picker, and pattern picker controls on windows. These types of button are specified as different button modes: kBMcolorpicker, kBMlinestylepicker, and kBMpatternpicker. When the user clicks on a button with one of these modes, OMNIS displays the appropriate picker palette. Note that the button does not generate an evClick event at this point. Instead, the button generates evClick after the user selects an item from the palette. The $event() method can obtain the selected value by using the $contents property of the pushbutton, which is a long integer. Pushbuttons with these button modes cannot have an icon, since the sample of the current color, pattern, or line style is displayed in the button.

Tab Pane Fields

Tab Pane fields have a number of panes on which you can place other fields. When the user clicks on a tab its associated pane is brought to the front displaying the objects on that pane. This type of field is useful for Options or Preference-style dialogs. The standard field properties control the overall size, position, and border style of the tabbed pane field, whereas each tab or pane has particular properties.



To add objects to a pane you have to click on the appropriate tab and drag the fields or background objects onto the pane. When you position an object over a pane, the border of the pane will highlight. To show any field or background object on all panes, right-click on the object and select the All Panes option from the context menu. You can move an object from one pane to another by dragging the object over the appropriate tab, pressing the Alt/Cmnd key to change the tab, repositioning the object on the new tab, and releasing the mouse.

The tab pane field type has several special properties under the General and Appearances tabs in the Property Manager. On the General tab, there is

tabcount	number of tabs or panes
currenttab	the tab or pane currently selected

In addition to the normal Appearance properties of a field, tab panes have the following; note that you can set the color and pattern of the area behind the tabs to blend in with your window background.

taborient	the position of the tabs: either at the top or bottom of the field
tabstyle	border style of the panes, a constant: kDefaultPanes, kSquarePanes, kRoundedPanes, kTrianglePanes
imagenoroom	when insufficient room shows just picture and not text for each tab
showimages	shows icons or pictures for each tab
showfocus	shows the focus for the selected tab
multirow	if true forces the tabs to stack rather than providing a scroll bar
forecolor	forecolor behind the tabs
backcolor	backcolor behind the tabs
backpattern	pattern for the area behind the tabs

You can set the following properties for each pane on the Pane tab of the Property Manager.

tabcaption	text or label for the tab
iconid	id of the icon for the tab: enable showimages for the tabbed field to display icons
tabtooltip	tooltip for the tab

Page Pane Fields

Page Pane fields are like Tab Panes but without the tabs. The **pagecount** and the **currentpage** property on the General tab in the Property Manager determine how many panes the field has, and which is currently visible for the placing of other fields. In design mode, you cannot click on the separate panes, so you need to set **currentpage** to bring a pane to the top. At runtime, you can use a method to set the **currentpage** property to bring a pane to the top. Like tab panes, page panes are useful for Options or Preference-style dialogs, or for creating your own wizards in which you need to step through a number of stages under the control of a method behind the field or window.

Tab Strip Fields

Tab Strip fields contain a number of tabs, only one of which can be selected at any time. You can add a method to a tab strip field that responds to whichever tab the user clicks, so it is very similar in operation to a set of radio buttons. You set the text and number of tabs for the tab strip by

entering a comma-separated list in the **tabs** property. For example, the text Bob,Mary,Fred will enable three tabs for the field with the specified text. The tab strip field type has several other special properties under the Appearance tab in the Property Manager.

backcolor	color in the area behind the tabs; turn off **ditherbackground** to get a solid color
selectedtabcolor	the color of the selected tab
tabcolor	the color of the tabs
selectedtabtextcolor	color of the text on the selected tab
tabtextcolor	the color of the text on the tabs
showedge	whether or not to show the edge of the tab strip
ditherbackground	whether or not to show the dithered background for the field
overlap	the overlap for the tabs in pixels
leftmargin	the indent for the left tab in pixels

Picture Fields

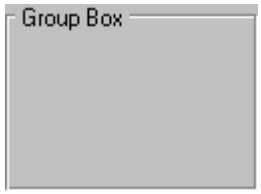
Many server databases let you store graphics, usually in binary fields or BLOBs. You can display these using an OMNIS picture field. You can paste any kind of graphical data into a picture field from the clipboard with Edit>>Paste, or you can read a Windows bitmap or metafile or a Mac PICT format file directly with the Edit>>Paste From File menu item. The focus for a picture field is shown as a dotted line around the field. Note that you can also link or embed pictures into your database using the OLE Picture external component.

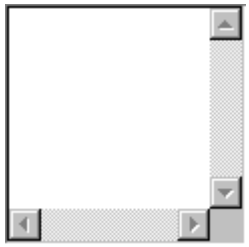
If you set the **sharedpictures** library preference OMNIS will store pictures in a proprietary format valid across all platforms supported in OMNIS. Storing pictures in shared picture format can result in slower displays and may increase disk storage. Once you put pictures in shared format, you can convert them back again by copying them to the clipboard. When you paste, the picture will appear in a standard graphics format.

The **noscale** property for picture fields is visible in design mode and in particular in the Property Manager. If set to kFalse (the default), pictures are scaled to fit the size of the picture field, otherwise if kTrue, pictures are not scaled retaining their aspect ratio.

Group Boxes and Scroll Boxes

Group boxes and scroll boxes let you group other fields on your window. They do not contain or display data themselves, they simply contain other fields and controls. You can create them from the Component Store and drag other fields within their borders.





You can edit the label for a group box in its **text** property. You can place any number of fields in a scroll box: this lets you place more fields on your window in a smaller confined area. Group and Scroll boxes can contain methods including a $event() method to detect events.

Shape Fields

Shape fields are graphic objects that have some general field properties, such as **visible, active**, and **enabled**. Therefore you can hide a shape field, make it inactive, or disable it just like an ordinary field. Shape fields can contain methods including a $event() method to detect events, so you can detect when the mouse enters or leaves the field. A shape field can be a rectangle, line, or text object.

Other Fields

List and grid fields, including combo boxes, dropdown lists, check lists, and popup lists, are described in the *Lists and Grids* chapter.

Popup menus are discussed in the *Menu Classes* chapter.

Subwindows, icon arrays, complex grids, data grids, string grids, headed list boxes, tree lists, modify report fields, and screen report fields are described in the *Window Programming* chapter in the *OMNIS Programming* manual.

Field Numbering and Tab Order

OMNIS assigns a number to each field as you place it on your window. This number is stored in the **order** property for the field. When you open a window and tab from field to field, OMNIS uses the order of these field numbers to guide it in moving to the next field regardless of the position of the field on the window. This lets you control how the user tabs or moves between fields. Note that background objects do not have a field number.

To show field numbers for the current design window

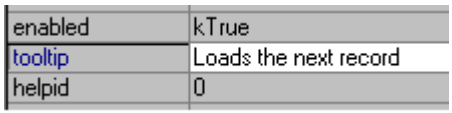
- Right-click on the window background
- Select the Show Field Numbers menu item from the window context menu

You can reorder the fields on your window, and hence change the tabbing order, by changing the **order** property of one of the fields: you can do this in the Property Manager for the field. When you change the order number for one field, other fields on your window will change too. OMNIS numbers the fields from top-to-bottom, and left-to-right, so when you change the number of a field, the other numbers will appear to shuffle or rotate.

You may have to experiment with the field numbers to achieve the tab order you want. When you open your window the cursor will move to the first field: the one numbered 1. Then as you press the tab key the cursor will move to each field in turn.

You can further control tabbing in the window using the **autotablen** property under the Action tab in the Property Manager. This property determines the number of characters you can type before OMNIS tabs to the next field automatically. For example, if you enter 4 for autotablen, the cursor will tab to the next field when you try to type a fifth character. The fifth character is ignored and the cursor jumps to the next field in the tab order.

Adding Tooltips to Window Objects

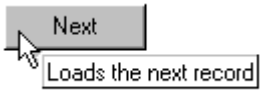


You can add tooltips to window fields and toolbar controls in the **tooltip** property for an object, and the **tabtooltip** property for each tab in a tab pane or tab strip field. Background objects cannot have tooltips. Note that you can use square bracket notation in the text for tooltips.

To enter a tooltip for an object

- Open your window and click on the object
- Open the Property Manager, or bring it to the top using F6/Cmnd-6
- Under the General tab select the **tooltip** property and enter a short help message

When the user positions the mouse over the object in runtime the help message pops up. For example, you can add tooltips to pushbuttons.

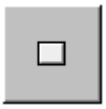


You can hide and show tooltips for all window objects and toolbars in OMNIS and your own libraries using the preferences **showwindowtips** and **showtoolbartips** control. These preferences are enabled by default, but you can change them using the Tools>>Options/Preferences menu item.

Background Objects

All types of window fields, that is, any window objects that potentially can hold data or receive events, are referred to as *foreground objects*. Any graphic objects you place on your window are considered to be *background objects*. The latter do not hold data, they are graphical devices for enhancing the appearance of your window.

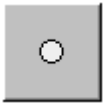
You can create various types of background objects including text and labels from the Component Store. To view background objects in the Component Store, click on the Background Objects button on the Component Store tool bar.



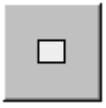
3D Rect
creates a 3d rectangle or square

Label
creates a field label

Line
creates a horizontal, vertical, or diagonal straight line



Oval
creates an oval or circle



Rectangle
creates a standard rectangle or square

Round Rectangle
creates a standard rectangle or square with rounded corners

Text
creates a text object

You place background objects on your window in exactly the same way as for fields. For example, you can click on a component icon in the Component Store and click-and-drag the

mouse on your window to create an object of a particular size and shape. If you hold down the Ctrl/Cmnd key as you draw an object, OMNIS constrains the objects as follows.

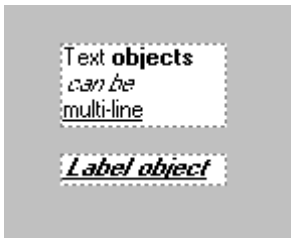
– rectangles and rounded rectangles draw as *squares*

– ovals draw as *circles*

– lines draw at *0, 45,* and *90* degrees only

Note that you can paste a picture or bitmap image into your window as a background object using the **Edit>>Paste** or **Edit>>Paste from File** menu items. The former choice pastes a picture you have cut or copied to the clipboard; the latter pastes a Windows metafile, bitmap or a Mac OS PICT file directly onto the window background.

Label and Text Objects



You can place text on a window (or report) using either a Label or Text object. These object types are represented by the constants kLabel and kText respectively. They have the same text properties under the Property Manager, but different usage: label objects do not support several features that are available for text objects. The main differences between label and text objects are as follows:

– Label objects *do not* wrap, and are therefore suitable for field labels or short single line text objects. Text objects do support multiple lines

– Label objects *do not* support the use of square bracket notation, whereas Text objects evaluate variables or calculations contained in [ ] placed in the text

– Label objects *do not* support rich text features within the text, whereas Text objects let you assign different character formatting (italic, bold, underline, and so on) to individual characters or words within a single text object

Background Object Properties

When you place a background object on your window the Property Manager opens showing the properties of the object. For Background Objects the Property Manager shows general, text, appearance, and data properties. These properties control the overall appearance of the object. Note that background objects do not have several of the properties of foreground objects such as **dataname, enabled,** or **visible**.

Note that once you have placed an object on your window you cannot change its type. For example, you cannot change a rectangle into a rounded rectangle. To create an object of another type you have to create a new one from the Component Store.

Generally, background objects are either *graphic* objects or *text* objects. All background objects have the same properties, but the text properties for graphic objects are irrelevant and are they are grayed out in the Property Manager. All background objects have a

particular size and position, set up using the **left, top, width**, and **height** properties. Also every background object has an **ident** or a number that uniquely identifies the object.

For text and labels you can change the **font, fontsize, fontstyle, textcolor**, and **alignment**. You change theses properties on the Text tab of the Property Manager. You enter the actual text for a text or label in the **text** property for the object. This property is grayed out for objects that do not contain text.

You can set the color and pattern of a background object under the Appearance tab in the Property Manager using the **backpattern, forecolor, backcolor**, and **bordercolor** properties, and you can set its border using **linestyle**. Furthermore, you can change the border effect of all background objects by setting the **effect** property. Objects can be inset, beveled, chiseled, shadowed, or various combinations of these.

External components

In addition to the built-in components within OMNIS Studio, you can extend the range of field types available to you by adding external components to the Component Store and integrating them into your applications. You add external components to a window class from the Component Store in the same way as the built-in fields and objects described in the previous sections.

OMNIS supports a number of different types of components.

– ActiveX controls
  pre-registered external components, available under Windows only

– Java Beans
  applets written in Java, currently available under the 32-bit Windows platforms only

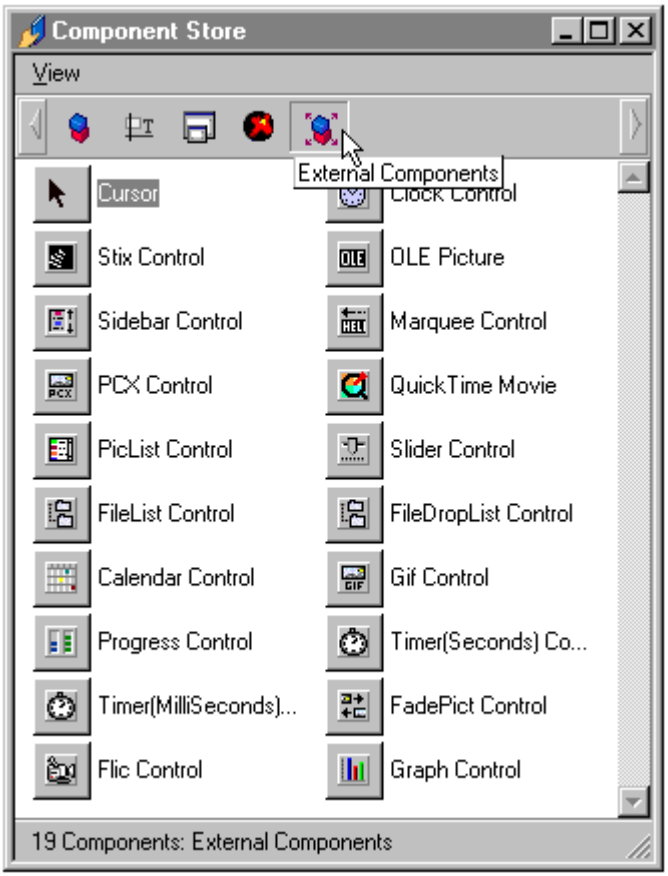
– C++ controls
  external components written in C++ that you can compile to run under any operating system supported in OMNIS

External component libraries are placed in the XCOMP folder under the main OMNIS folder and viewed in the Component Store. To use external components on a 68K Macintosh you need the Code Fragment Manager.

To use an external component, its library must be loaded. A number of components are automatically loaded the first time you start up a copy of OMNIS Studio, and you can decide which others you need. All the external components currently loaded are available under the External Components button in the Component Store toolbar; this button is visible when you have a window or report design screen on top and at least one component is currently loaded.

To use external components

- Open your window or report class in design mode
- Open the Component Store, or bring it to the top, by pressing F3/Cmnd-3
- Click on the External Components button in the Component Store toolbar



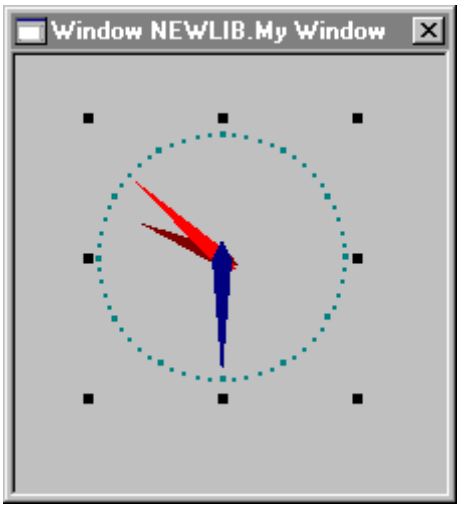
To place an external component on your window

- Click on the required external component in the Component Store
- Click and drag on your window to create a field

or you can

- Double-click on an external component in the Component Store to create a field automatically
- Size and reposition the external control field

For example, you can click on the Clock Control and create a clock component on your window. The Property Manager displays an external component's General field properties, as well as its own properties on the Custom tab (or the tab has the component's name).



Clock Control shown on a window

Property Manager showing properties of the Clock Component

On the General tab, each external component has the **componentlib** and **componentctrl** properties representing the name of the component library and the name of the current control field. Some controls have an **about** property which, when clicked in the Property Manager or called using the notation, shows an About window for the component.

Controls have their own custom properties, methods, events, and constants and you will need to refer to the vendor documentation for information on how these work. The events and constants for a control are listed in the Catalog, whereas the methods are listed in the Property Manager under the Methods tab. For example, the QuickTime Control lists its methods under the Methods tab in the Catalog.

Showing External Components in the Component Store

The external components initially visible in the Component Store are preloaded, but you can show other components that may already be available. You can also load or register other external components that reside in your system.

All the external components currently available in OMNIS are listed in the system table called #EXTCOMPLIBS. You can access this system table from the IDE Browser or from the Component Store.

To show an external component in the Component Store

- Select the Library>>External Components menu option on the IDE Browser menu bar

or

- Open a window or report design screen and make sure it's the top window
- Select View>>External Components from the Component Store menu bar

or

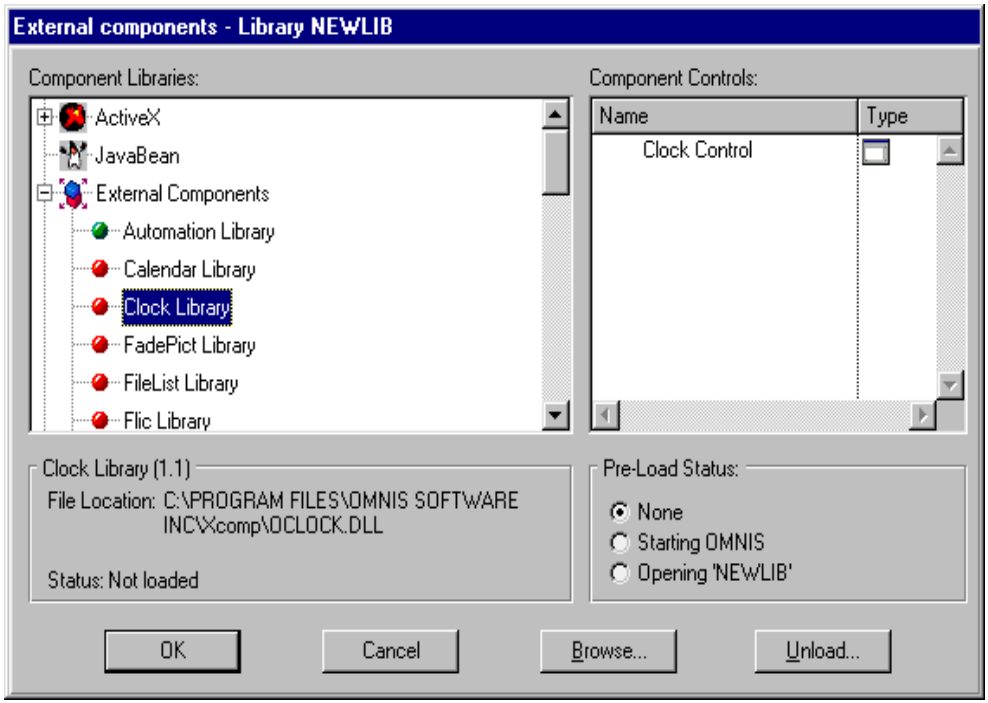
- Right-click on the Component Store and select External Components from the context menu

or

- Open the IDE Browser, and show the system tables in your library using the Browser Options (press F7/Cmnd-7 when the Browser is on top)
- Double-click on the system table #EXTCOMPLIBS

External Components Dialog

The External Components dialog contains a tree list that includes all the external components currently available on your system. External component libraries that are currently loaded are shown with a green dot, those not loaded are shown with a red dot. A library may contain one or more controls within it; normally it would be logical for these controls to be related in some way. In addition to visual external components, the library may also contain external objects, which are described in *OMNIS Programming*.



The dialog lists all the different types of external components and external objects including ActiveX, JavaBeans, and background component types. If you select a Component library in the left hand pane, its Component controls and their type, such as whether they can be placed on a window or report are shown for each control in the right hand pane. The type is indicates by icon(s). You control the availability or preloaded status of each library by clicking on one of the radio buttons as follows

– **None**
the component is not loaded, but when used on a window or report it is loaded

– **Starting OMNIS**
the component is always preloaded in OMNIS and is available to all OMNIS libraries

– **Opening ‘**the current library’
the component is preloaded in the specified library when it is opened

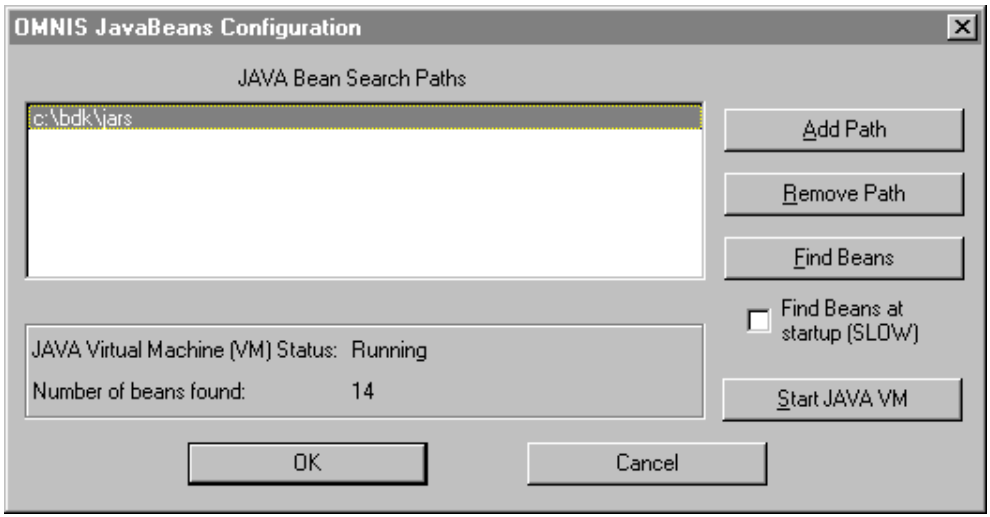
Note that you have to close and reopen the dialog to refresh the list. Certain components, such as the Graph and OLE components, are loaded by default. Setting an external component library to be always loaded in OMNIS will increase the time OMNIS takes to startup, whereas setting a component to load on opening a library or window will slow down the opening of the library or window only.

If you are experiencing problems with loading or using an external component, it may be that there is some non-OMNIS software that you need to install. The Trace log, opened in the debugger under Options>>Open Trace Log, will show what, if anything, is missing.

Loading or Registering External Components

You can load external components not currently listed in the External Components dialog, including ActiveX and Java components, using the Browse button.

To use Java Beans you must install the Java Development Kit (JDK) version 1.1 or later. You should also download the Bean Development Kit (BDK), since this is a prime source of Java Bean examples. Java Beans can reside anywhere on your system, therefore OMNIS needs to be aware of where they are. To find the beans on your system, use the Find Beans button on the Java Beans Configuration dialog.



Note that this process may take some time, since your whole system has to be scanned.

ActiveX controls can reside anywhere on your system, and are registered in the registry, or under Windows 3.1, in the OMNIS.INI file. You must *register* an ActiveX control before you can use it in OMNIS. When you *unregister* one, it is no longer available in OMNIS.

To load or register an external component

- Open the External Components dialog, as above, and click on the Browse button
- Select the Component type, for example, ActiveX or OMNIS External component

For ActiveX components

- Click on the Register button
- Navigate the open dialog, select the ActiveX component, and click on Open
- You will have to restart OMNIS to view ActiveX components

For OMNIS External components

- In the open dialog go to the XCOMP folder under the main OMNIS folder
- Select the component and click on Open

To unload an external component

- Select the external component in the External Components dialog
- Click on the Unload button

Graphs

The Component Store contains a Graph external component that lets you create many different types of graph including bar charts, pie charts, and stock market graphs. The Graph component is described in the separate *OMNIS Graphs* manual.

Background External Components

In addition to the controls already described, OMNIS Studio supports background external components. These background or graphical components, such as the Wash Control and Tile Control examples, are shown under the Background Components button in the Component Store. You can also create and integrate your own background external components into your applications.

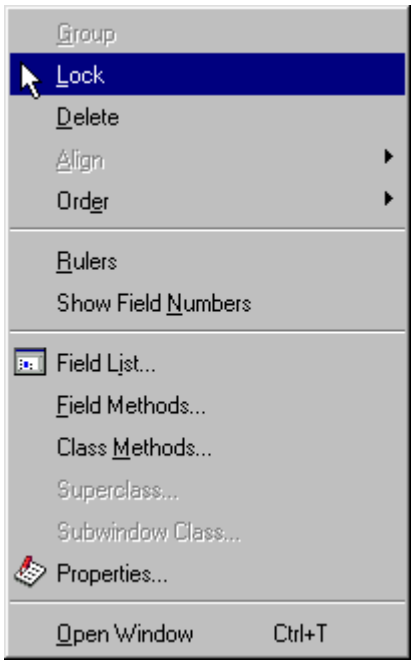
Writing your own External Components

You can write your own C++ external components and add them to your applications as you would other external components. If you want to do this, you may find it useful to look at the source of the example external components provided in OMNIS. The source is found on the OMNIS website which you can access from the OMNIS Help. The example components vary considerably in complexity, but you should start by looking at the simpler ones, such as the Generic externals. Full details of how to create your own external components is documented in the *OMNIS Studio External Components* manual available on the OMNIS website (www.omnis-software.com).

If you have created any external components of your own to run under OMNIS Studio version 1.x, you must recompile them for OMNIS Studio 2.0.

Modifying Windows and Fields

When creating a window in design mode you can modify it as much and as often as you like: you can carry on adding fields and background objects, you can change the style and properties of the window, you can delete fields and objects, or you can change the properties of fields at any time.



Once you have placed objects on your window you can move them, resize them, align them, group them, copy and paste them, and you can even drag an object onto another window, or drag objects into the current window.

While you are modifying your window you can Right-click on a field or the window background to open a context menu that lets you modify the object or group of objects under the mouse. This menu contains options that apply to the current object, the selected group of objects, or the current window class. The options are:

– **Group**
joins selected objects together so you can select or move them as a group, and places a check mark against the option; you can ungroup a selected group by unchecking this option, but note you cannot ungroup a locked group

– **Lock**
locks the size and position of an object or group of objects and places a check mark against the option; you can unlock an object or group by unchecking this option

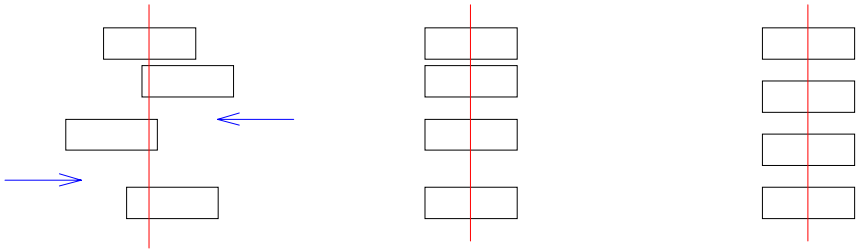
– **Delete**
deletes the object or currently selected group of objects

– **Align**
opens the align submenu that lets you align objects: you can align objects according to their left, top, right, or bottom edges; you can make objects the same width or height; you can center objects horizontally or vertically; and you can evenly distribute or space objects horizontally or vertically

To arrange a number of vertically-oriented objects

- Select the objects and Right-click inside one of the objects
- Select the Align>>Center Vertically option
- Open the context menu again and select Align>>Distribute Vertically

Center vertically... **Distribute vertically...** **Gives you...**



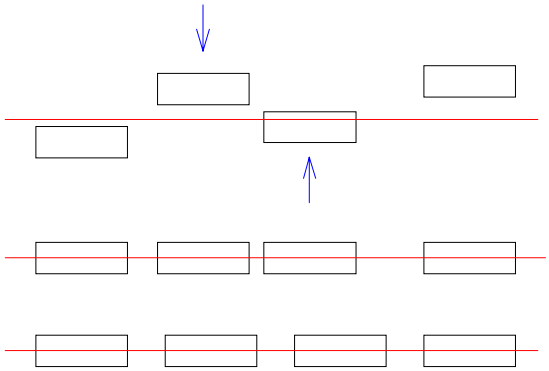
To arrange a number of horizontally-oriented objects

- Select the objects and Right-click inside one of the objects
- Select the Align>>Center Horizontally option
- Open the context menu again and select Align>>Distribute Horizontally

Center horizontally...

Distribute horizontally...

Gives you...

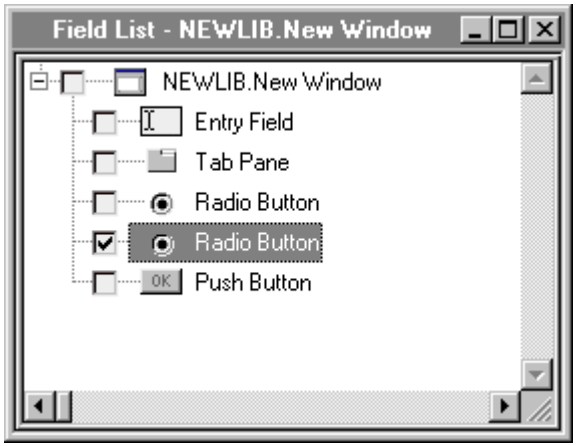


– **Order**
opens the order submenu that controls the layering of overlapping fields and objects: you can send an object to the back or bring it to the front, or you can send an object back one layer or bring it forward one layer

– **Rulers**
adds rulers to the current window; the rulers have the units set in the **usecms** OMNIS property under Tools>>Options/Preferences (set it to false for inches)

– **Show Field Numbers**
shows or hides field numbers on the current window: note that background objects do not have field numbers; see below for details about field ordering

– **Field List**
opens a list of fields for the current window



The field list displays all the fields and other objects on the current window. You can expand the node for a container field, such as a tab pane or scroll box, to view its contents. The field list indicates the currently selected field with a check mark; you can select a field in your window by checking a field in the list.

– **Field Methods**
opens the field methods for the currently selected field; the option is grayed when you click on the window background, or an object that cannot contain methods

– **Class Methods**
opens the class methods for the window

– **Superclass**
opens the superclass for the currently selected inherited field or object; the option is grayed if the window does not have a superclass

– **Subwindow Class**
opens the window class for the currently selected subwindow field; the option is grayed for all field types except subwindow fields

– **Properties**
displays the properties for the window or currently selected object

– **Open Window**
toggles between the design window and the open window: note that you can press Ctrl/Cmnd-T to *test* a window whenever it is on top; this keypress toggles a window between design and runtime mode

Moving and Sizing Objects

When creating or modifying your window you can rearrange objects by moving and resizing them with the mouse or arrow keys. You can click inside an object or Shift-click on a number of objects to select them as a group.

To duplicate an object under Windows, you hold down the Ctrl key and click and drag inside the object, or under MacOS you Option-drag the object. Holding the Shift key down as well constrains the movement to vertical or horizontal, or you can use a design grid in the window.

You can cut or copy and paste objects on your window using the Edit menu or its equivalent keys, and you can delete objects altogether by pressing Backspace.

You can also move and size your window at any time. When you are designing a window, its position and size on the screen reflect its actual position and size when you open it for data entry (although you can override this using programming). You can drag the title bar of your window to move it, and you can click and drag the edge or corner of the window (assuming it has a size box or border) to resize your window.

Using the Grid

To help you arrange the objects on your window you can use a design grid, which is a property of the window itself.

To set the properties of the window grid

- Open the Property Manager for your window by clicking on the background
- Click on the General tab (you may need to scroll down to see the grid properties)

With these properties you can show the window design grid (shown as a series of dots), size and align objects to the grid, and set the fineness of the grid. You can set **horzgrid** and **vertgrid** to any number from 2 to 200 pixels. With very fine settings such as 2 and 3 pixels, OMNIS displays every other grid node, but objects are still sized and aligned to the current setting.

showgrid	kFalse
aligntogrid	kFalse
sizetogrid	kFalse
horzgrid	8
vertgrid	8

When you enable the grid by setting the **aligntogrid** or **sizetogrid** properties OMNIS will not reposition and resize existing objects on your window automatically: the grid will size and align any new objects you create after you enable the grid, but you can resize and re-align existing objects by hand.

Window and Field Methods

You can create a window class and add fields and other objects to the window from the Component Store, but to make your window properly function you need to add some programming behind your window. To do this, you write code that accesses the standard methods in the window class. You add *class methods* to the window itself to control the window behavior and handle events for the window, and you add *field methods* to each field or control on your window to control the behavior of the field and handle its events.

You can add up to 501 methods to each field or control on your window, and a further 501 methods to your window class. You enter these methods using the method editor.

When you create a window from the Component Store it contains a $construct() and $destruct() method by default. You can add code to these methods that control the opening and closing of the window class. Most window fields from the Component Store have an $event() method that contains event handling code for that type of object. For example, a standard entry field has code that detects when the user enters and leaves the field, and a tab pane field has code that detects which tab has been selected. You can add further event handling code to the $event() method or add other methods to the same field.

To add a method to a window class

- Open your window class
- Right-click on the window background to open the window context menu
- Select the Class methods option

You can add class methods to your *window* that

– control the window when it is opened or closed

– respond to user clicks and tabs

– detect when another window is brought to the front

To add a method to a window field

- Open your window class
- Right-click on the field to open a context menu
- Select the Field methods option

You can add methods to each *field* that detect

- when the user *enters* the field
- when the user *leaves* the field or presses the tab key
- when the user clicks on a pushbutton or other control
- when the user's mouse passes over the field
- when the user clicks on a line in a list field

For further details about programming methods and handling events, see the *Methods and Notation* and the *Events and Messages* chapters in the *OMNIS Programming* manual.

Chapter 7—Menu Classes

Menus let end users perform standard operations in your application, such as enter data or print reports. The definition for a standard menu is stored in a *menu class*. You can create your own custom menus and install them on the main application menu bar using the *Install menu* command, on the menu bar of a window, or as a popup or context menu on a window. You can create hierarchical menus that drop down off another menu, and you can incorporate standard OMNIS menus such as File and Edit into your application.

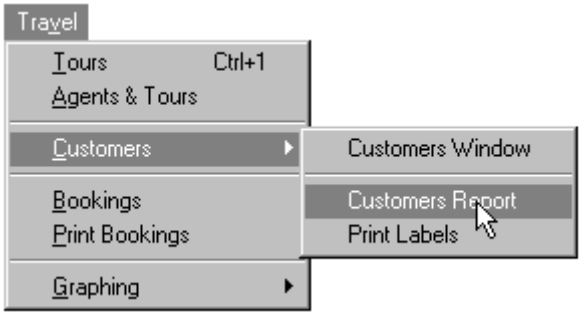
This chapter describes the different types of menu available in OMNIS and how you create them using a menu wizard or from scratch. It also describes menu properties and how you modify them to change the behavior of your menus.

Menu Types

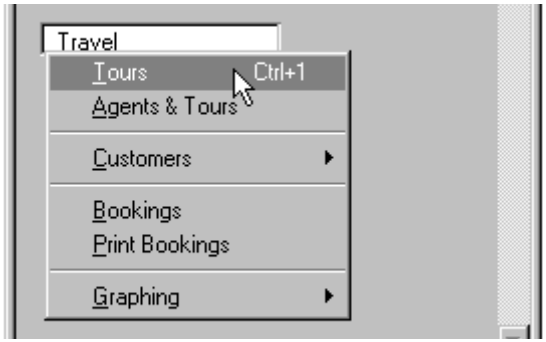
The types of menu classes you can create are:

- **Standard dropdown menus** you can install any menu class on the main menu bar; you can add shortcut keys and control access to menus using user levels, you can check and uncheck individual menu lines and enable/disable them.

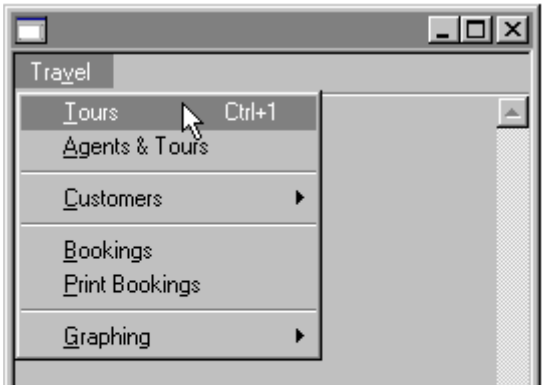
- **Hierarchical menus** you create a hierarchical menu as a separate menu class and add it to another menu line; when the user selects the line a menu drops down



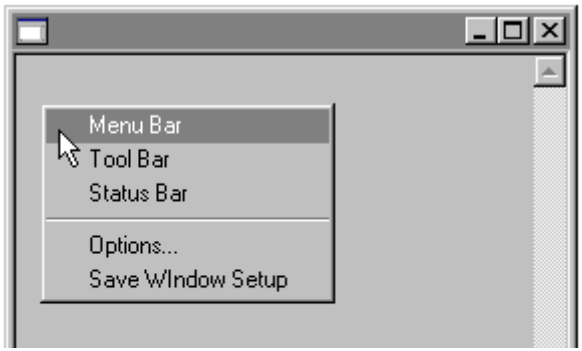
- **Popup menus**
  this type of window field pops up a standard menu when you click on it



- **Window menus**
  you can install any standard or custom menu on the menu bar in a window



- **Context menus**
  you can define a context menu that pops up when you Right-click on a field or window



You can add up to 500 lines or menu items to a menu class, but in practice you will only need the first twenty-or-so for most types of menus. You can add a keyboard alternative, or shortcut key, to each menu line when you create the class.

Methods do the real work behind the menu. You can add methods to the class itself and each menu line. The class methods can initialize the menu when it is installed, and the line methods could do anything from open a window, print a report or series of labels, or insert a

row into your database. When you select a line in the installed menu, OMNIS runs the method behind that menu line.

Creating Menus using Wizards

This section describes how you can create a menu class automatically, using a wizard from the Component Store. A menu you create in this way lets you open window classes and print reports. You can also add hierarchical menus to your new menu using the standard wizard. To make full use of the standard menu wizard you should create your window and report classes first. The following wizards are available

Menu Wizard
creates a menu containing menu lines to open window classes and print report classes; can also contain hierarchical menus

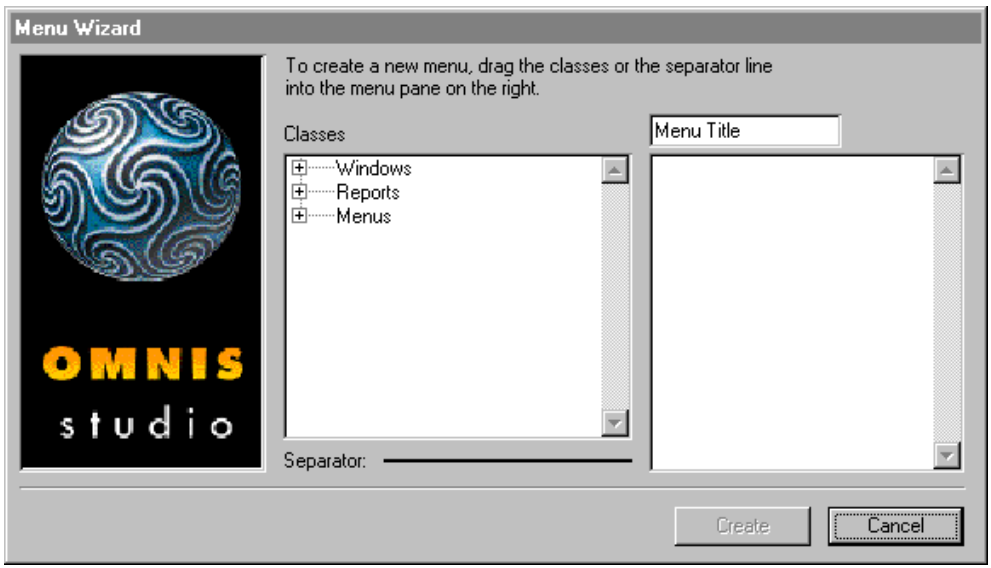
File Menu Template
creates a menu that you can use to replace the standard File menu; you can edit this menu class and add your own menu lines

To create a menu class using a wizard

- Open your library in the Browser
- If you like, you can display the classes in your library using the View>>Down One Level menu option on the Browser menu bar
- Open the Component Store or bring it to the top using F3/Cmnd-3
- Click on the Menu Classes button in the Component Store toolbar to show all the menu templates and wizards

- Drag the Menu Wizard from the Component Store onto your library in the Browser

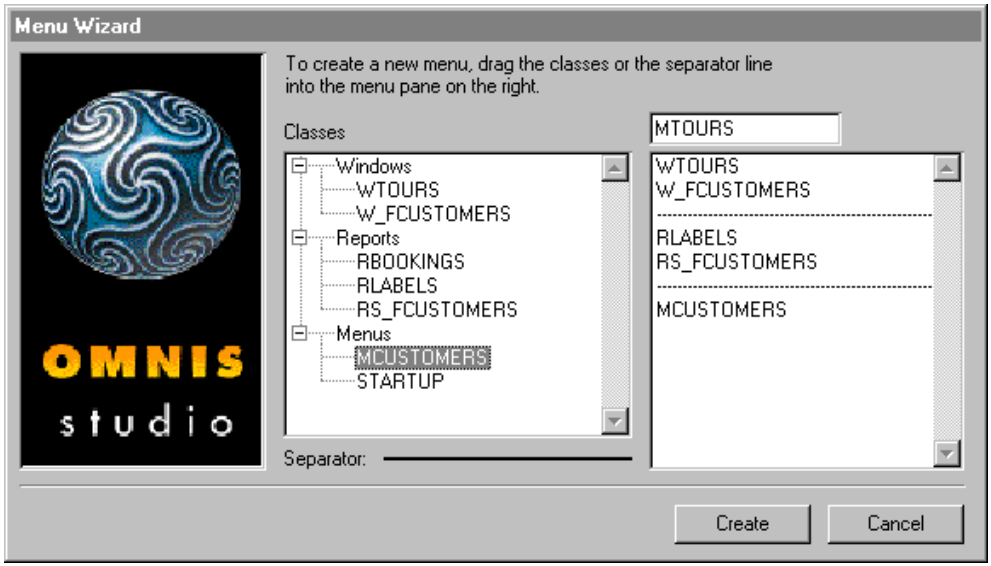
- Name the new menu class and press Return, or click in the Browser



The Menu Wizard displays all the available window, report, and menu classes in your library and lets you select which classes you want to include on your menu. You can expand each group and drag individual classes or separators onto the menu pane, or you can drag a whole group of classes onto the menu pane.

If you click Cancel at any time, the wizard is halted and the new class is removed from your library. To continue

- Drag individual window, report, and menu classes onto the menu pane, or drag a group name onto the menu pane to include all classes in that group
- Drag separators onto the menu pane and drop them in between the classes
- If necessary, you can drag classes or separators in the menu pane on the right to rearrange your menu
- Edit the menu title; this is the name that will install on the main menu bar



- Finally, click on Create

When you finish in the Menu Wizard the new menu class is opened ready for you to modify or install. To modify your menu you need to edit its properties in the Property Manager, but you can edit the text for a menu line by clicking on the line and typing some new text. You can also add new menu lines, and you can add methods to the menu class or the lines in the menu. These are all described in this and subsequent chapters.

To install your menu

- When the menu design screen is on top you can Right-click on the menu class, and select Install Menu from the context menu

or from the Browser you can

- Right-click on the new menu class, and select Install Menu from the context menu

Default Menu Template or Wizard

The *default menu template* or wizard is the one used when you create a new class using the Class>>New>>Menu option in the Browser, and it's also the class shown under the Default Classes button in the Component Store toolbar. The default menu template appears under the Menu Classes button in the Component Store, shown by a check mark on the template icon. However, you can change the default template for menu classes.

To change the default menu template or wizard

- In the Component Store toolbar, click on the Menu Classes button
- Right-click on the class or wizard you want to make the default
- Select the Make Default option from the context menu

For example, you can make the Menu Wizard the default template and from thereon, when you use Class>>New>>Menu in the Browser, the Menu Wizard will be called by default. When you click on the Default Classes button in the Component Store, the template or wizard you set as the default is shown. The remainder of this chapter assumes that the class called "New Menu" is the default class.

Creating a New Menu

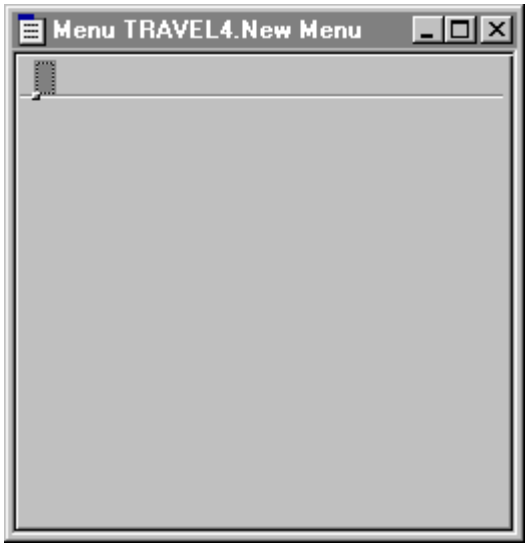
This section describes how you can create a menu class from the Component Store or from the Browser using the New Menu template. A menu you create in this way contains no menu lines or methods, but it lets you create your menu entirely from scratch. To design a menu you create a menu class, specify the text you want to appear in each menu line, and add the methods to each line in your menu using the method editor.

To create a new menu class

- Open your library in the Browser
- Display the classes in your library using the View>>Down One Level menu option on the Browser menu bar
- Drag the template called "New Menu" from the Component Store onto the Browser

or

- From the Browser menu bar select Class>>New>>Menu
- Name the new menu
- Double-click on the new menu class to modify it



The new menu does not contain any menu lines or methods. The cursor is at the top-left of the menu editor. You create the menu title, menu lines, and separators using the keyboard.

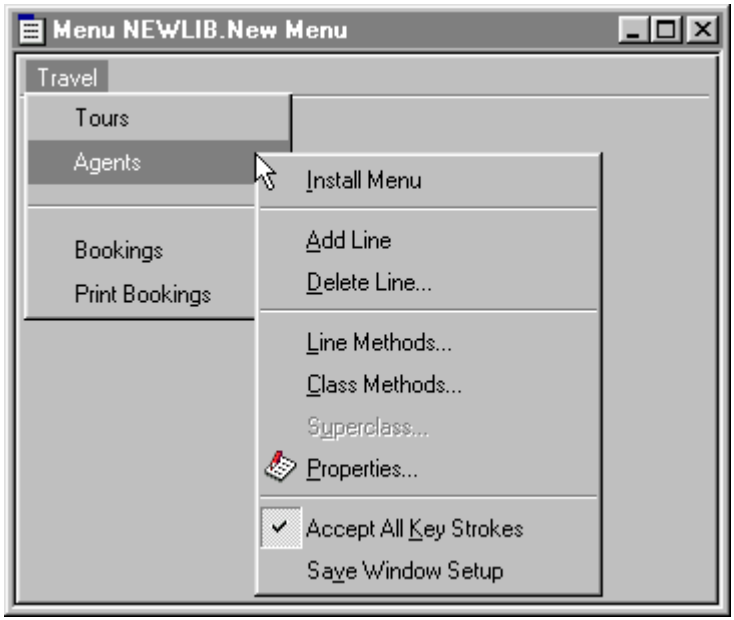
To add a menu title and menu lines

- Type the menu title; this is added at the top of the menu editor
- Press Return once to go to the first line in your menu
- Type the text you want to appear in the first menu line
- Press Return once to go to the next line in your menu
- Continue adding menu lines using Return to go to the next line

To create a menu line separator

- Press the Return key twice

You can use drag and drop to re-order the lines in a menu. You can also drag a menu line from one menu class to another to copy the line. You can add and delete lines by Right-clicking on your menu and selecting Add or Delete Line from the context menu.



You can also use this context menu to install your menu on the menu bar, add methods to the class, and examine the properties of the menu. The Install Menu option adds your menu to the main menu bar. You can click on your menu and see how it looks, but until you add methods behind each line in your menu class it will not do anything.

Menu Line and Class Methods

You can create a menu class and add each menu line, but to make your menu properly function you need to add some programming behind your menu. To do this, you write code that accesses the standard methods in the menu class. You can add *class methods* to the menu itself to control the menu when it is installed. And you can add *line methods* to each line in your menu: a line method is executed when the corresponding menu line is selected in the installed menu.

You can add up to 501 methods to each line in your menu, and a further 501 methods to your menu class. You enter the methods for lines and classes using the method editor. You can open the methods for a menu line by double-clicking in the margin to the left of the menu text; double-clicking on the menu text lets you edit the text.

When you create a menu from the Component Store it contains a $construct() and $destruct() method by default. You can add code to these methods that control the installing and closing of the menu. In addition, each menu line has an $event() method in which you add the code you want to run when the menu line is selected. For example you could use the *Open window instance* command in a line method to open a window, or you could use the

Print report command to print a report to the current destination. A menu line method can do literally anything you want it to do using an OMNIS command or series of commands.

To add a method to a menu line

- Open your menu class
- Right-click on the appropriate menu line to open a context menu
- Select the Line methods option

To add a method to a menu class

- Open your menu class
- Right-click on the background of the menu editor to open the menu context menu
- Select the Class methods option

Menus and Code Classes

You can put general-purpose methods that you are likely to need throughout your application into a code class, and call these methods from the line methods in your menu class using the Do code method command. You can call the same methods from the toolbars in your library too, which saves duplicating methods. For further details about writing methods, see the *Methods and Notation* chapter in the *OMNIS Programming* manual.

Menu Properties

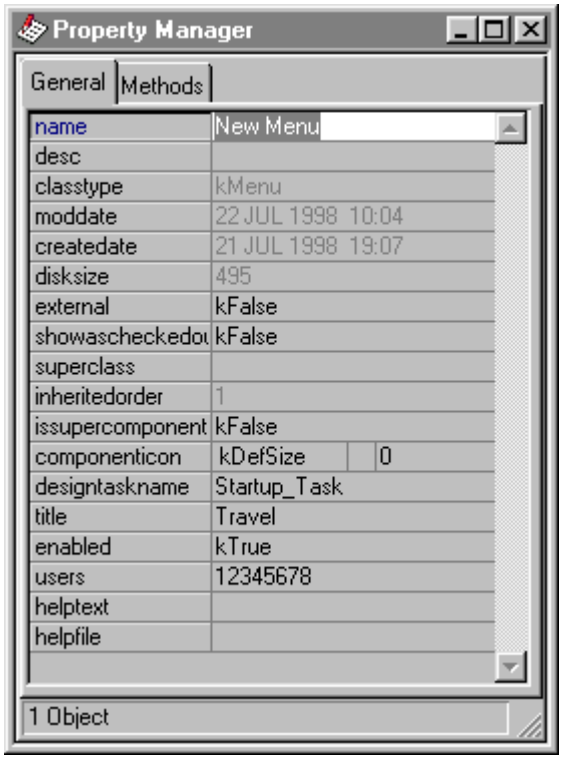
This section describes menu class and menu line properties, and how you modify these to change the appearance and behavior of your menu. Many of the standard functions of a menu, such as enabling or disabling a menu line, adding shortcut keys, or setting passwords for each menu option, are properties of each menu line.

To view the properties of a menu class

- Open the menu class in design mode
- Click on its title

or

- In the Browser, Right-click on the menu class and select the Properties menu item



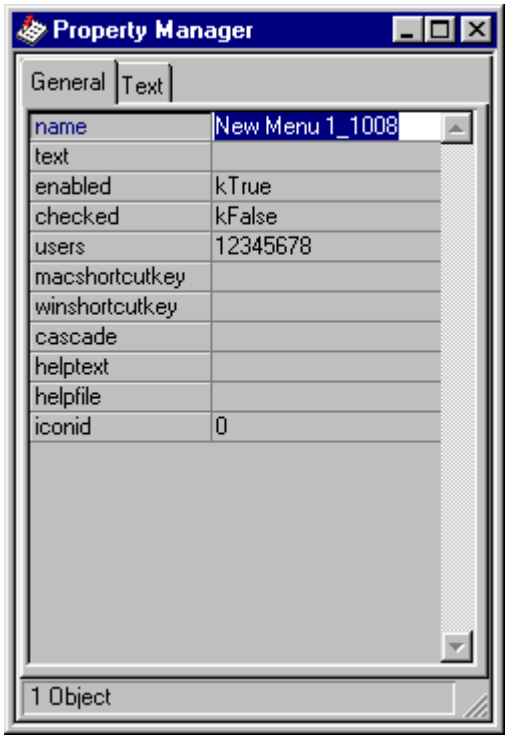
The general properties of a menu class are as follows: **title** is the name that appears on the menu bar, **enabled** controls whether the whole menu is accessible or not, and **users** controls which users are allowed to access the menu (see below for details about menu access). For details about the other class properties such as **name** and **classtype**, and setting **designtaskname**, see the *Libraries and Classes* chapter; help for menus is described later in this chapter.

To view the properties of a menu line

- Open the menu class in design mode
- Click on the menu line

or

- Right-click on the menu line and select the Properties menu item



The Property Manager opens showing the properties for the selected menu line. You can set the following properties for each menu line or separator.

name	the object name of the menu line, that is, the internal name that uniquely identifies this menu line
text	the text for the menu line; you can enter this text directly in the menu editor or here in the Property Manager
enabled	whether the menu line is enabled or not: if you set this to false, the line will be grayed out in the installed menu
checked	whether the menu line is checked or not: normally you check and uncheck a line in response to a user action using a method
users	controls which users have access to the menu line; see below for details about access
macshortcutkey	the shortcut key character for the menu line under MacOS
winshortcutkey	the shortcut key character for the menu line under Windows
cascade	name of the menu class attached to this menu line as a hierarchical menu

helptext	the help message displayed in the status help bar
helpfile	help file for context-sensitive help
iconid	id of the icon for the menu line; if empty the menu line has no icon

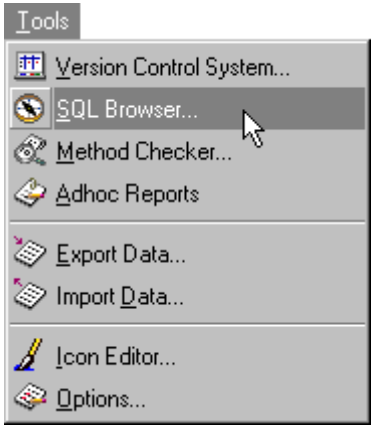
In addition to the general properties, under MacOS you can set the **fontstyle** property under the Text tab to make a menu line bold, italic, or underline and you also have the Outline and Shadow font styles.

Menu Icons

The OMNIS IDE menus and your own custom menus can have icons for each menu line. Menu icons appear under Windows 95 and NT only, provided the $root preference $style97 is set to kTrue.

Menu lines in your own custom menus have the $iconid property in which you can specify the id of a 16x16 icon for the menu line; larger icons are not available for menu lines. If the property is empty (the default) the menu line does not have an icon.

If a menu line can be checked, for example, it is an option that can be on or off, OMNIS uses the checked and unchecked state of the icon provided the icon has multiple states in the icon data file.



Shortcut Keys

You can specify a shortcut key or keyboard alternative for each line in your menu. When the end-user presses the specified key combination the menu line is activated. Under Windows you can add Ctrl and Alt key combinations to menu lines. Under MacOS you can add Cmnd and Option key alternatives. You can further modify keyboard alternatives with the Shift key under any OS. You enter these keys in the Property Manager for the menu line, or by pressing the required key combination when the appropriate menu line is selected in the menu editor.

The menu editor context menu has the option Accept All Key Strokes. When checked (the default) the menu editor accepts all keystrokes, including shortcut keys, and enters them into the current menu line. When this option is unchecked you cannot enter menu lines directly from the keyboard, in this case you have to enter the text and shortcut key for each menu line in the Property Manager.

To add a shortcut key in the menu editor

- Open your menu class in design mode
- Select the menu line and press the key combination you want to assign to it

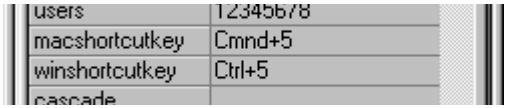
For example, select the menu line and press Ctrl or Cmnd and the number key “5” to add the Ctrl/Cmnd-5 shortcut key, or press Ctrl or Cmnd and the letter “A” to add the Ctrl/Cmnd-A shortcut key to the current menu line. Whichever platform you are using, the appropriate shortcut key is entered for all platforms automatically.

Certain shortcut keys cannot be inserted in this way, because they have functionality that is detected and intercepted by OMNIS or the operating system. They are Ctrl/Cmnd-T and Ctrl/Cmnd-S on all platforms, plus Cmnd-Q and Cmnd-W under MacOS.

To add a shortcut key in the Property Manager

- Right-click on the menu line and select the Properties option
- Click on the droplist in the **winshortcutkey** property
- In the dialog that pops up, press the shortcut key combination you want to assign to the menu line and close the dialog
- Repeat for the **macshortcutkey** property, if required

The shortcut key properties should contain the appropriate control keys and letter key.



You should avoid using standard key combinations that appear in OMNIS or your operating system. MacOS function keys on extended keyboards activate the menu option with the corresponding Cmnd-number combination. Thus, F1 is the same as Cmnd-1. You cannot use the Shift-Cmnd-n options, where n is a digit from 0-9, because the MacOS uses these options.

Furthermore you should make sure that no two menu items in a menu have the same shortcut key and that no two menus installed at the same time have the same shortcut key. Duplicates will be unpredictable depending on which menus are installed at the time or the order in which they appear on the main menu bar.

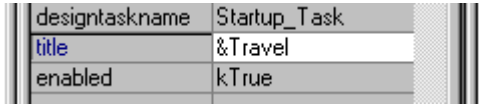
Alt Shortcuts Keys under Windows

Under Windows you can add Alt key equivalents to menu lines and to the menu title itself. You specify the key by including an ampersand ("&") before the character in the menu line or menu title. For example, if you want your users to open a menu called Travel with the Alt-T key combination, add an ampersand before the T in the menu title. In this case, the text for the menu title should be “&Travel”. Likewise you can add an Alt shortcut key to any letter in a menu line. For example, to add the Alt-S shortcut key to a “Customers” menu line, the text for the menu line should be “Cu&stomers”.

You can include the ampersand in the appropriate menu line or title when you enter the item in the menu class editor, or you can add it to the text or title property for the item in the Property Manager. Usually you add shortcut keys as an afterthought, in which case it is easier to do it in the Property Manager. Note you cannot select the menu title or line and press the required Alt-key combination to assign this type of shortcut key: you have to enter it directly into the menu editor when you enter the line or using the Property Manager.

To add a shortcut key to a menu title using the Property Manager

- Right-click on the menu title and select the Properties option
- In the Property Manager select the **title** property
- Add the “&” character before the letter you want to activate

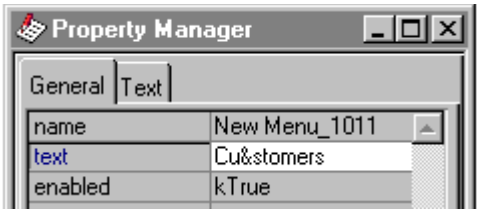


To add a shortcut key to a menu line using the Property Manager

- Right-click on the menu line and select the Properties option
- In the Property Manager select the **text** property for the menu line
- Add the “&” character before the letter you want to activate

or

- Simply click in the appropriate line in your menu and add the “&” character



The following screen shots show Alt key shortcuts for a menu title and menu line and the resulting installed menu.

This menu class...

gives you this installed menu...

If you want to use the “&” character as a part of a menu title or menu line, you must insert the ampersand twice. For example, the menu name "&Clients && Calls" would produce the menu title "Clients & Calls" with the shortcut key Alt-C.

You should avoid certain Alt key combinations that are used in standard menus in OMNIS or your operating system, such as Alt-F, which opens the File menu.

Hierarchical menus

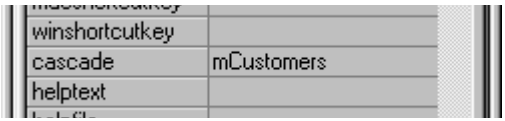
A *hierarchical menu* is a menu that drops down from another menu line when you select the option. You can create a hierarchical menu using any previously defined menu class. For example, you can create a menu that includes a number of related options, and add its class name to the **cascade** property for the menu line in your main menu.

To add a hierarchical menu to a menu line

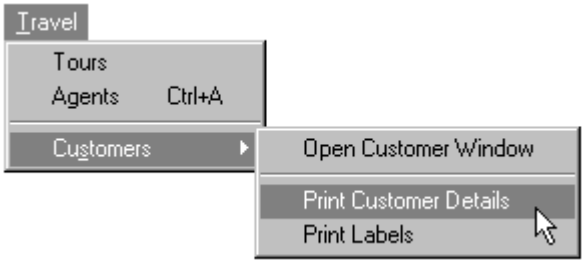
- Create a separate menu class for your hierarchical menu in the menu class editor, described earlier in this chapter
- Open your main menu, that is, the one you want to add the hierarchical menu to
- Select the line where you want to add your hierarchical menu, and click on the **cascade** property in the Property Manager
- Enter the name of your hierarchical menu class

For example, you could create a menu class called mCustomers and add to it various options that relate to customers.

Then enter its class name in the **cascade** menu line property in your main menu.



When you add a hierarchical menu to a menu line, OMNIS places an arrow against the menu line in the menu editor. This also appears in the installed menu indicating there is a hierarchical menu attached to this menu line. When you select the option in the installed menu the hierarchical menu drops down.



You can cascade menus up to five levels deep under MacOS, and up to eight levels under all Windows platforms. You should avoid cascading a menu off itself, or creating a chain of menus that cascade off each other recursively.

You can create multiple instances of a cascading menu using the menuname/* notation in the $cascading property for the menu line. Otherwise if you specify a simple menuname for the $cascading property and create multiple instances of the parent menu, you get only one instance of the cascading menu.

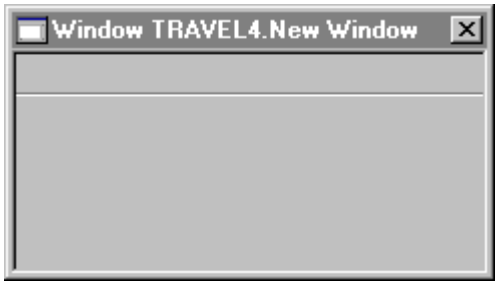
Window Menus

A *window menu* is a menu installed on the menu bar of a window. A window menu bar is a property of the window class itself. To show the menu bar for a window you must enable the **hasmenus** property. Having enabled the menu bar for a window, you can drag menus from the Browser and drop them onto a window menu bar.

Note that many of the OMNIS commands and methods that apply to standard menus, such as *Install menu*, do not apply to window menus.

To enable a window menu bar

- Open the window in design mode
- Click on the background of the window to show its properties, or press F6/Cmnd-6 to bring the Property Manager to the top
- Under the General tab enable the **hasmenus** property
- Under the Appearance tab set the **menuedge** property



When you enable the window menu bar all the objects on your window including fields and background objects will move down. You can add any type of menu to a window menu bar, including your own custom menus or the standard OMNIS menus.

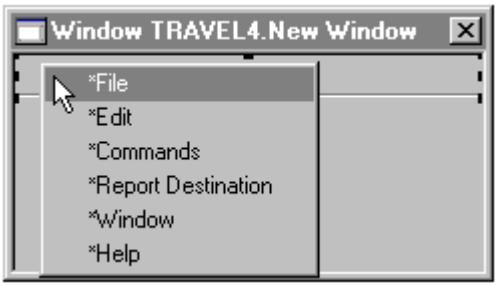
To add your own menu to a window menu bar

- Enable the menu bar for your window, as above
- Locate the Browser containing the classes in your library
- Drag your menu class from the Browser onto the window menu bar
- Release the mouse where you want to place your menu

Your menu is added to the window menu bar, but is grayed out in design mode. To try out your window menu, open the window and click on the menu.

To add an OMNIS menu to a window menu bar

- Enable the menu bar for your window, as above
- Right-click on the window menu bar and select the menu you require



To remove a menu from a window menu bar

- In design mode, drag it out of the window menu bar and release the mouse

Popup Menus

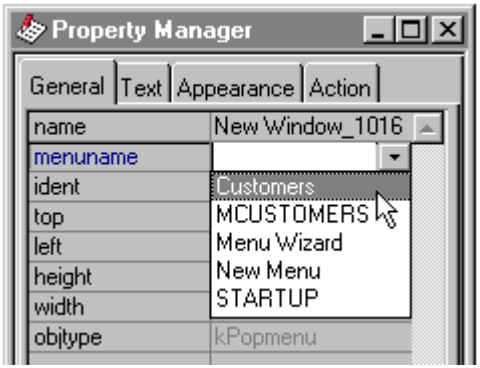
A *popup menu* is a type of window field that opens a menu when you click on the field. You can create a popup menu using any previously defined menu class and you can add any of the standard OMNIS menus such as File and Edit to your window as popup menus. When you create a popup menu field you enter the name of the menu class in the field's **menuname** property.

You can use the constant kDefaultBorder for the $effect property to ensure the menu has the default border style for the current operating system.

All its other properties are the same as any normal window field. You set up the properties of the menu itself in the menu class, as described for standard menus.

To create a popup menu field

- Open your window in design mode
- Drag a Popup menu field onto your window from the Component Store
- Locate the Property Manager, or press F6/Cmnd-6 to bring it to the top
- Select the **menuname** property for the field
- Enter the name of your menu class, or select it from the dropdown list



When you open your window and click on the popup menu field, your menu drops down under the window field.

Context Menus

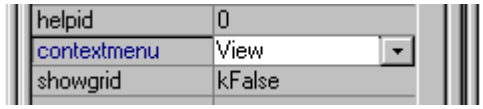
A *context menu* is a menu that pops up when you Right-click on the background of an open window or a field; under MacOS you Ctrl-click to popup a context menu. Context menus appear throughout the OMNIS design environment to help you access methods and so on, but you can add context menus to any of the windows in your application. To create a context menu you enter the name of the menu class in the **contextmenu** property for the window or field. You set up the contents and properties of the context menu itself in the specified menu class, as described earlier in this chapter.

To create a context menu for a window

- Create the menu class for your context menu

For example you could create your own View menu for a particular window: such a menu could show and hide the window menu bar, tool bar, status bar, and so on.

- Open your window in design mode and click on its background to open the Property Manager
- Select the **contextmenu** property and enter the name of your menu class or select its name from the droplist



- Open your window and Right-click on the window background to try out the context menu

When a context menu pops up an instance of the menu is created and its $construct() method is called. Therefore you should be careful what code you put in this method.

Lists, Headed lists, and Icon arrays can have a second context menu. This is stored in the $contextmenu property, in the form Menu1[,Menu2]. If specified, Menu2 applies to clicks in the white space in the list, whereas Menu1 is opened in response to clicks on list lines or icons. If the second menu is not specified, Menu1 is opened for all clicks.

The evOpenContextMenu event has an additional parameter, pClickedField, which contains an item reference to the field or window instance that has been right-clicked. Menu instances have an additional property called $contextobj. For context menus, this is an item reference to the field or window instance that has been right-clicked to bring up the menu.

Passwords and Menu Access

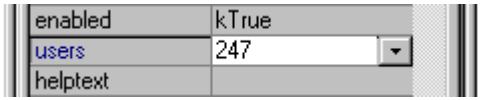
You can restrict access to certain parts of your library including menu items by setting up a system of passwords. You can define varying degrees of access for up to eight users or passwords, plus a master password. Several different users can use the same password: you are not limited to literally eight users. You can use the passwords set up in your library to control access to the menus in your library. For a description of setting up passwords see the *Library Tools* chapter.

Access to menus or menu lines is set in the **users** property for the menu item and utilizes the user numbers and passwords set up in your library. The default is to allow access to all the menu items in your library for all users or passwords, that is, passwords 1 to 8. The master password has access to all menus at all times. The users property contains the string “12345678”. To restrict access to a menu or menu line, you delete the user number from the users property. For example, to restrict access for user 4, delete the number 4, which leaves the string “1235678”. To allow access to a menu item, you include the user number in the users property for the menu or menu line. For example, to allow access for user 4 only, delete the default string and enter the number 4 only.

To set access for a menu

- Open your menu in design mode, or locate it the Browser
- Right-click on your menu and select the Properties option
- In the **users** property, enter the user number or numbers you want to allow access for

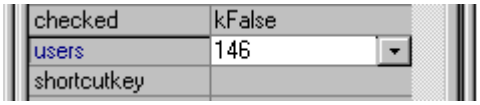
For example, to allow access to this menu for users 2, 4, and 7, enter the numbers 2, 4, and 7. In effect, this disables access to your menu for users 1, 3, 5, 6, and 8.



To set access for a menu line

- Open your menu in design mode, and select the appropriate menu line
- Press F6/Cmnd-6 to open the Property Manager, or Right-click on the menu line and select the Properties option
- In the **users** property, enter the user number or numbers you want to allow access for

For example, to allow access to this menu line for users 1, 4, and 6, enter the numbers 1, 4, and 6. In effect, this disables access to this particular menu line for users 2, 3, 5, 7, and 8.



When the user opens your library using a password that denies access to a particular menu line, the line is grayed out.

If you restrict access to a line containing a hierarchical menu, access is restricted to the hierarchical menu too. If you restrict access to a menu title, the whole menu is grayed out and the user cannot access the menu.

Note that if you restrict access to one particular user, you do not restrict access to any other users, that is, password access is not hierarchical. For example if you deny access for user 5 you do not restrict access for users 6, 7 and 8 automatically.

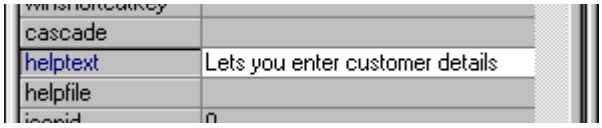
The master user is the only user that can change passwords, that is, you have to use the master password (if there is one) when you enter your library to change the passwords in that library. Passwords are not assigned in new libraries, therefore you are the master user by default.

Status Bar Help for Menus

You can add short help messages to menus and menu items that display on the window status bar or the main OMNIS help bar. You enter the message in the **helptext** property for the menu title or menu line. The **hasstatusbar** window property enables the status bar for a window class, and the **helpbaron** OMNIS preference enables the main OMNIS help bar. You can change the font and point size for the main OMNIS help bar with the **helpfont** property. Under MacOS you can display the menu help text in Help balloons by enabling the **balloonson** property. Note that you can set the OMNIS preferences under the Tools>>Options/Preferences menu item.

To enter a menu help message

- Open your menu and click on a menu line or the title
- Open the Property Manager, or bring it to the top, using F6/Cmnd-6
- Under the General tab select the **helptext** property and enter a short help message



When you install your menu and position the mouse over the menu option, the help message is displayed in the window status bar for window menus, and in the main OMNIS help bar for menus installed on the main menu bar.

Chapter 8—Toolbar Classes

You can create your own toolbars that contain buttons and other controls that lets the user access common options and functions in your application. In OMNIS you define a toolbar as a *toolbar class*. This chapter describes how you create toolbars in your own library. The standard OMNIS toolbars are described in the *OMNIS Tools* chapter.

You can install your own custom toolbars in the top, left, right, or bottom docking area of the main OMNIS application window using the *Install toolbar* command. Toolbars can also be floating. You can add toolbars to any window class too, but you must create your toolbar class first before you can add it to a window. The toolbar commands, such as *Install toolbar*, refer to toolbars in the main docking areas, not window toolbars.

Toolbar classes can contain methods and variables. As with menus, the methods you place behind each control do the real work in a toolbar. You can add methods to the toolbar class itself and to each control. When you click on a control in your toolbar, OMNIS runs the $event() method behind the tool.

A method behind a control could do anything from open a window, print a report or series of labels, or insert a row into your database. In line with usual GUI design practice, you can put the methods to do these things into a code class and access them from your toolbar, *and* access the same methods from a menu.

The Component Store contains a New Toolbar class that is the default toolbar class. This template class does not contain any controls or methods, but lets you create your toolbar from scratch. The Component Store also contains a number of other template toolbars, under the Toolbar Classes button, that you can use as a basis for your own toolbars. They include a File, Standard, and View template which you can use in a window toolbar: the methods behind these toolbars call code in the current window.

Creating a New Toolbar

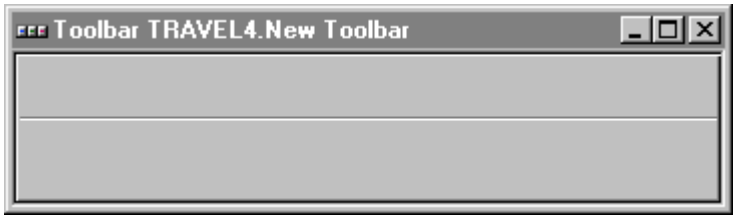
To add a toolbar to your library, first you need to create a toolbar class and add the various buttons and controls you want on your toolbar. Next you add the methods to each tool using the method editor. You can create a toolbar class from the Component Store or from the Browser using Class>>New>>Toolbar.

To create a new toolbar

- Open your library in the Browser
- Display the classes in your library using the View>>Down One Level menu option on the Browser menu bar
- Drag the template called “New Toolbar” from the Component Store onto the Browser

or

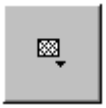
- From the Browser menu bar select Class>>New>>Toolbar
- Name the toolbar class
- Double-click on the toolbar class to modify it



Toolbar Controls

You can add many different types of controls to your toolbar, including buttons, lists, menus, check boxes, radio buttons, and various style and color pickers. When you open a toolbar class in design mode the Component Store displays the different types of tools.

Control	Description
	**Separator** adds a space between toolbar controls
	**Pushbutton** a push button control that runs a method when you click on it
	**Dropdown list** list control that lets you select a value from a dropdown list
	**Combo box** list control that lets you enter a value or select one from a list
	**Popup list** list control that lets you select a value from a popup list
	**Popup menu** menu control that lets you select an option from a popup menu
	**Radio button** toggle control that lets you select one of several mutually exclusive choices
	**Check box** toggle control representing on/off , yes/no, 1/0 status

Control	Description
	**Font list** list that lets the user pick a font for an object
	**Font size list** list that lets the user select the font size for an object
	**Line style picker** palette that lets the user pick a line style for an object
	**Color picker** palette that lets the user pick a color for an object
	**Pattern picker** palette that lets the user pick a pattern for an object

To add a toolbar control

- Open the Component Store, or bring it to the top using F3/Cmnd-3
- Drag a control from the Component Store and drop it onto your toolbar

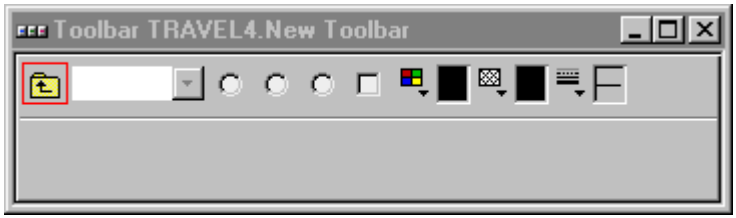
or

- Double-click on a control in the Component Store to add it to the toolbar; this adds a control to the right of the selected control or at the beginning of your toolbar if no control is selected

When you drop a control onto your toolbar class it snaps into position. You can add as many buttons and controls as you like, and you can use the separator control to put space between groups of tools. You can drop a control anywhere within the active area or between existing controls. Once you've placed controls in a particular order, you can drag them to the left or to the right to reposition them on your toolbar.

Dragging a control from one toolbar class to another copies the control to the destination class.

The following screen shot shows a toolbar with some of the different types of controls, including button, droplist, radio buttons, check box, color and pattern pickers.



You can click on check boxes and radio buttons to turn them on or off. If you place several radio buttons together, without separators, they behave as a group. That is, when you select one radio button in a group the currently selected one will be deselected. Popup lists and menus look like buttons, but when you click on them in the installed toolbar a list or menu drops down.

To remove a toolbar control

- Open your toolbar class in design mode
- Click on the control and press the Delete key, or you can press Backspace to delete the control to the left of the currently selected one

At this stage some of your tools may not have icons or text labels, and when you install the toolbar the controls will not do anything when you click on them. All of these features you add in the properties and methods for each control.

Toolbar Properties

The properties of a toolbar class set the default initial docking area for the installed toolbar, whether or not it can be dragged out of a docking area, and whether or not you can resize the toolbar when it is floating. In addition, you can set the user access for the toolbar in its class properties.

To view the properties of a toolbar

- Click on the toolbar class in the Browser, and press F6/Cmnd-6 to open or bring the Property Manager to the top

or

- Open your toolbar class in design mode
- Click on the background of the toolbar (make sure a control is *not* selected)
- Press F6/Cmnd-6 to open the Property Manager

title this is the name displayed in the title of the toolbar when it is floating (if it is allowed to float)

allowdrag if true lets you drag the installed toolbar out of the docking area: when undocked the toolbar is said to be *floating*

allowresize if true lets you resize the toolbar when it is floating

initialdockingarea the docking area into which the toolbar is installed initially: you can install a toolbar into the top, bottom, left, or right docking area, or you can make the toolbar floating

enabled if true the toolbar is enabled: if you disable a toolbar it is grayed out and the controls do nothing

users determines which users have access to the toolbar; see the *Library Tools* chapter for details about passwords and access

helpfile the name and partial path of the help file for the toolbar

Tool Properties

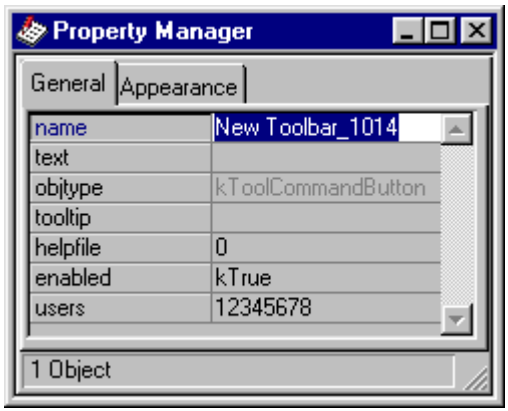
You set the appearance and behavior of each tool or control on your toolbar in the Property Manager. Note that you *cannot* change the **objtype** of an existing toolbar control.

To view the properties of a toolbar control

- Open your toolbar class in design mode, and click on the control

or

- Right-click on the control and select the Properties option to open the Property Manager

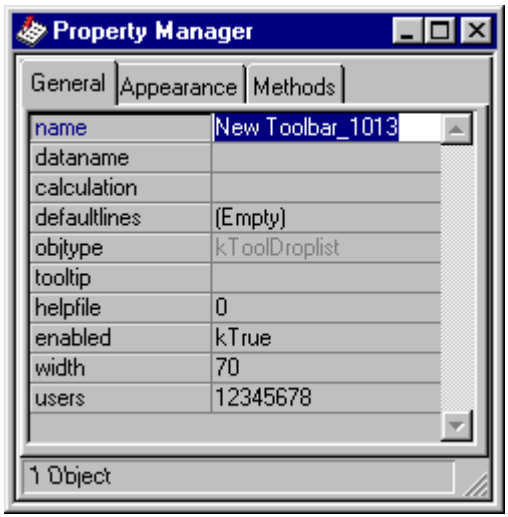


name	the object name of the control: this can be anything you like but should be unique within the toolbar class
text	the text displayed with the control
objtype	the control type: note you cannot change the type of an existing control
tooltip	the tooltip text for the control
helpfile	help file name for context-sensitive help for the control
checked	if true the toggle control is initially checked, for check boxes and radio buttons only
enabled	if true the control is enabled: if you disable a control it is grayed out and does nothing when selected
width	the width of the control in pixels; for list type controls only
users	determines which users have access to the control; see the *Library tools* chapter for details about passwords and access

Note that control separators have only the **name** and **objtype** properties.

Combo Box, Droplist, and Popup List Properties

Combo boxes, droplists, and popup lists have all the general properties of a toolbar control in addition to special properties including **dataname**. You can also specify the **width** of the various list type controls in pixels.



For droplists you specify the name of your list variable in the **dataname** property and you need to supply a **calculation** to format the list lines for display. Alternatively, you can enter a number of **defaultlines** for the droplist.

For combo boxes you specify the **listname** and **calculation** for the list part of the control, and **dataname** for the entry field part of the control. You can enter a number of **defaultlines** that will always popup in your combo box.

For popup lists you must specify the name of your list variable in the **dataname** property and you need to supply a **calculation** to format the list lines for display. Note you cannot enter default lines for a popup list.

Popup Menu Properties

For popup menu controls you must specify the name of the menu class to appear in your toolbar in the **menuname** property.

Font List Properties

For the Font Style and Font Size controls you specify the **listname** and **calculation** for the list part of the control, and the **dataname** for the entry part. A range of fonts and sizes appear in the controls by default, but you can enter your own fonts and sizes in a list or in the **defaultlines** property. At runtime, the font style or size selected by the user is returned

as the **contents** of the entry field part of the control. These pickers can be used with the Modify Report Field described in the *Window Programming* chapter in the OMNIS Programming manual.

Line, Color, and Pattern Pickers

The Line, Color, and Pattern pickers have all the general properties of a toolbar control as well as the **text** and **iconid** properties. The standard line, color, and pattern palettes appear in these controls by default. At runtime, the value selected by the user is returned as the **contents** of the control. These can be used with the Modify Report Field described in the *Window Programming* chapter in the *OMNIS Programming* manual.

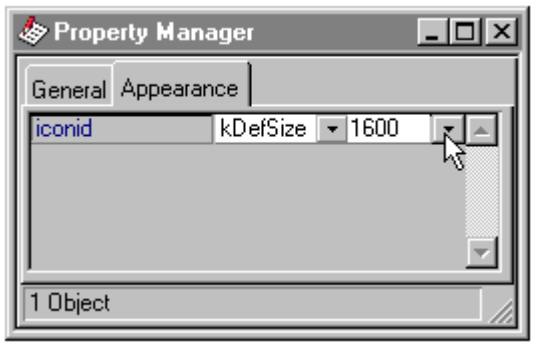
Toolbar Icons

You can add an icon to most types of controls, all except the list and combo types. You can specify an icon for a control in its **iconid** property. Icons for tool controls are located in the OMNISPIC icon data file, but you can add your own icons to the USERPIC icon data file and use them in your toolbars. See the *Library Tools* chapter for details about creating your own icons.

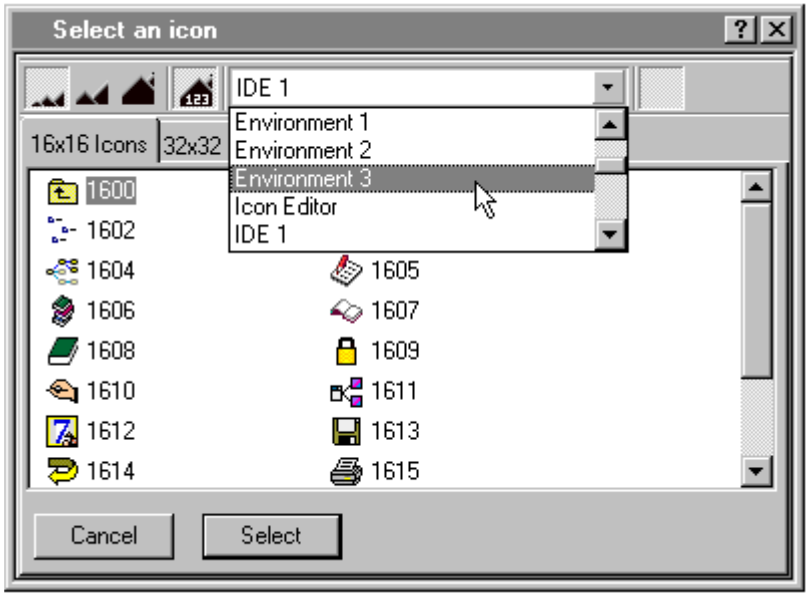
Controls that you can check or uncheck, such as radio buttons and check boxes, display different icons for the different checked and unchecked states. All possible states for these controls are stored in the OMNISPIC icon data file.

To add an icon to a toolbar button

- Open your toolbar class in design mode
- Click on the button to view the Property Manager
- Click on the Appearance tab in the Property Manager
- Click on the right-hand droplist in the **iconid** property



When you click on the id droplist for the **iconid** property, the Select an icon dialog opens in which you can select an icon for the control. You can select a different group of icons using the dropdown list of the icon selector toolbar.



You should use 16x16 icons for toolbar buttons, and you can use 32x32 icons for buttons that display text. 48x48 icons do not display for toolbar buttons.

- Select the icon you want and click the Select button, or double-click on the icon

Tooltips

You can add tooltips to individual toolbar controls in the **tooltip** property for the control. To hide and show tooltips for the toolbars in OMNIS and your libraries you can set the **showtoolbartips** OMNIS Preference, available under the Tools>>Options/Preferences menu item.

To enter a tooltip for a control

- Open your toolbar class in design mode, and click on the control to open the Property Manager

or

- Right-click on the control and select the Properties option

- Under the General tab of the Property Manager select the **tooltip** property and enter a short help message for the control

Tool and Class Methods

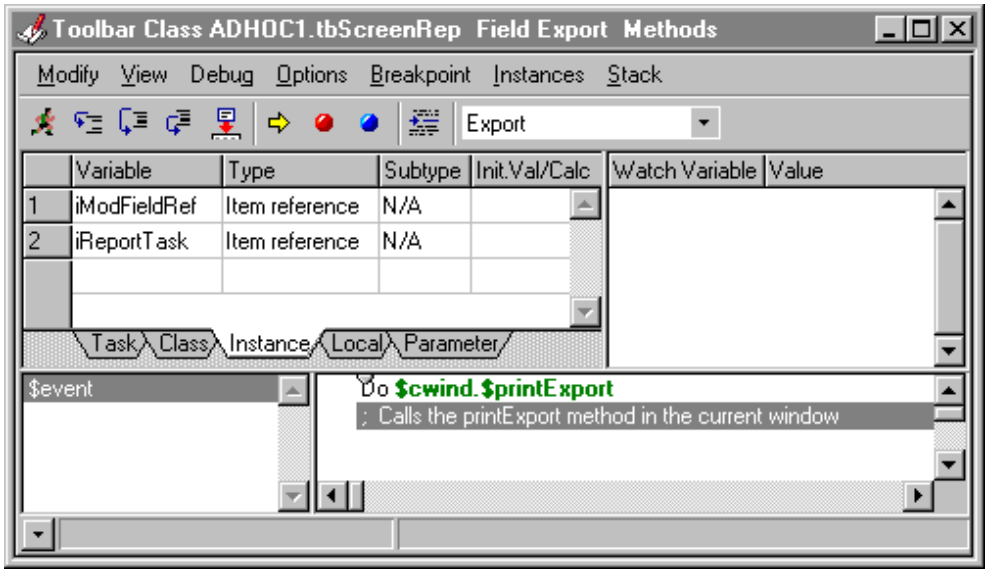
You can create a toolbar class and add many different types on control, but to make your toolbar properly function you need to add some programming “behind” each control on your toolbar. You can add *Class methods* to the toolbar itself to control the toolbar when it is opened, and you can add *Tool methods* to the controls in your toolbar: a tool method is executed when the corresponding tool or control is clicked on in the installed toolbar.

You can add up to 501 methods to each tool or control in your toolbar, and a further 501 methods to your toolbar class. You enter the methods for tools and toolbar classes using the method editor.

When you create a toolbar from the Component Store it contains a $construct() and $destruct() method by default. You can add code to these methods that control the installing and closing of the toolbar. In addition, all controls except separators have a $event() method in which you add the code you want to run when the tool or control is clicked on. For example, you could use the *Open window instance* command in a tool method to open a window, or you could use the *Print report* command to print a report to the current destination. A tool method can do literally anything you want it to do using the OMNIS commands or the notation.

To add a tool method

- Open your toolbar class
- Right-click on the appropriate tool to open a context menu
- Select the Tool Methods option



To add a method to a toolbar class

- Open your toolbar class
- Right-click on the background of the toolbar editor to open the toolbar context menu
- Select the Class Methods option

Toolbars and Code Classes

You can put general-purpose methods that you are likely to need throughout your application into a code class, and call these methods from the tool methods in your toolbar class using the *Do code method* command. You can call the same methods from the menus in your library too, which saves duplicating methods. For further details about writing methods, see the *Methods and Notation* chapter in the *OMNIS Programming* manual.

Installing Toolbars

You can install a toolbar class at any time from the toolbar editor itself; this is useful if you want to see how the toolbar looks while you're designing it. However, in your finished application you can use the *Install toolbar* command or the notation to install a toolbar. You can also add any toolbar class to the docking area of a window using the **toolbarpos** property.

To install a toolbar from the toolbar editor

- Open your toolbar class in design mode
- Right-click on the background of your toolbar
- Select the Install toolbar option from the context menu

The **initialdockingarea** property of a toolbar class determines which docking area the toolbar is installed into. Toolbars install into the top docking area of the main OMNIS application window by default.

If you have enabled the **allowdrag** property for toolbar class you can drag the installed toolbar out of the docking area; the toolbar is now *floating*. If you have enabled the **allowresize** property in the toolbar class you can resize the floating toolbar.

You can Right-click on a docking area and select the Show Text option from the context menu to show the text for each toolbar control.

Docking Areas

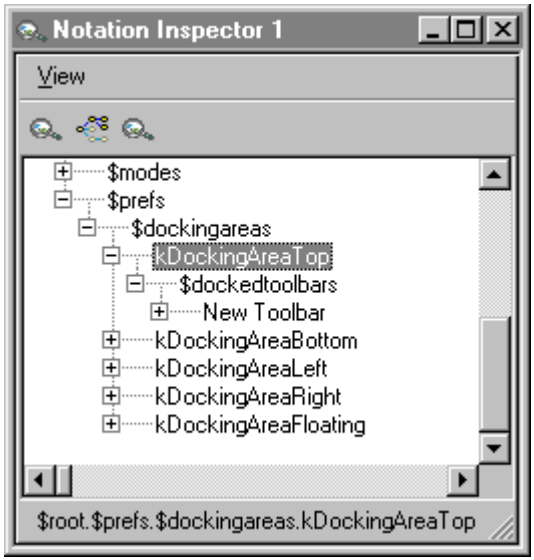
You can install a toolbar into the top, bottom, left, or right docking area in the main OMNIS application window. Note that most list-type controls, such as the Dropdown list and Combo box types, are not displayed in a toolbar if it is installed into the left or right docking area. They display as expected when the toolbar is at the top or bottom, or is floating.

You can right-click on a docking area to open its context menu which lets you show and hide the text labels for any installed toolbars. You can show text for the IDE toolbars as well as your own custom menus.

You can view and change the properties of the main docking areas using the Notation Inspector and the Property Manager.

To view the docking areas in the Notation Inspector

- Press F4/Cmnd-4 to open the Notation Inspector
- In the Notation Inspector, expand $root and the $prefs group
- Expand the $dockingareas group



The $dockingareas group contains the main docking areas in the OMNIS application window. Each docking area has a group called $dockedtoolbars containing all the currently installed toolbars.

To change the properties of a docking area

- In the Notation Inspector, click on a docking area: for example, click on kDockingAreaTop to examine or change the properties of the top docking area
- Press F6/Cmnd-6 to open the Property Manager, or bring it to the top

Each docking area has the $visible, $allowdrop, and $showtext properties. The latter displays the text labels for controls stored in the toolbar class.

Chapter 9—Report Classes

You can create many different types of report using a template or wizard, each with very different layouts and data handling capabilities. With reports you can print out all or a subset of your data, and collect up data from different sources and print it on a single report. Each type of report in your application is defined as a *report class*. This chapter describes report classes and their properties.

Reports can contain data fields, pictures, text, and graphics. You can also place graphs on your reports, or base a report on an OMNIS list. You can print reports to a number of destinations, including the current printer, the screen, a file, a port, or the clipboard. *Modify report fields* and *Screen report fields* let you view and modify reports in a window class and are described in the *Window Programming* chapter in the *OMNIS Programming* manual.

You use report fields and sections to build all types of report. You can use standard *data fields* that can contain data from your server or OMNIS database. You can use *picture fields* to display picture data. You place s*ections*, or horizontal dividers, across your report class that structure and position the data in the printed output. You can create subtotals, totals, header and footer sections for most types of reports. By setting the appropriate properties in a report class you can print labels as well. Furthermore you can add methods to a report class and the fields and section markers on the report.

In addition to report classes in your library file, you can create a special type of report, called an *ad hoc report*. You can use ad hoc reports to query data on your server or OMNIS database. They are stored as separate files on disk, and can be created, opened, and modified by end users. Ad hoc reports are described at the end of this chapter.

Creating Reports using Wizards

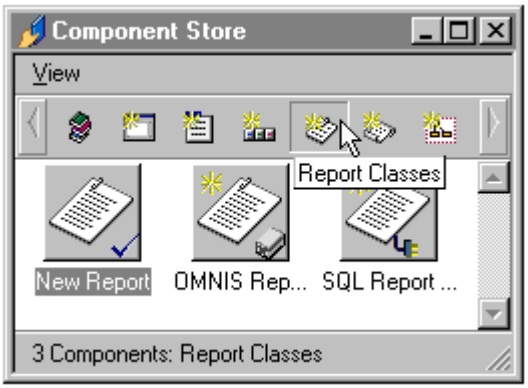
This section describes how you can create a report class automatically, using a wizard from the Component Store. A report you create in this way contains a number of fields that map directly to table or file class fields in your library, which lets you print data on your server or OMNIS database. Before you can use report wizards you must create the schema, table, or file classes necessary for SQL or OMNIS data access; this is described in the *Data Classes* chapter. The following wizards are available

SQL Report Wizard
creates a report based on a table class; each separate field on the new report maps to a schema column, which in turn maps to your server database

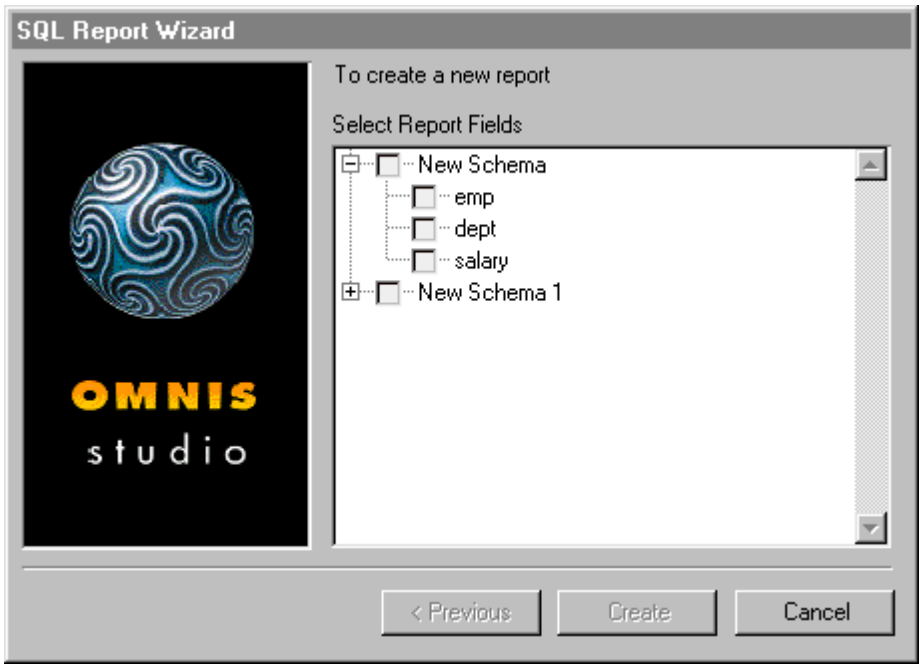
OMNIS Report Wizard
creates a report based on a file class; each separate field on the new report class maps to a file class field

To create a report class using a wizard

- Open your library in the Browser
- If you like, you can display the classes in your library using the View>>Down One Level menu option on the Browser menu bar
- Open the Component Store or bring it to the top using F3/Cmnd-3
- Click on the Report Classes button in the Component Store toolbar to show all the report templates and wizards



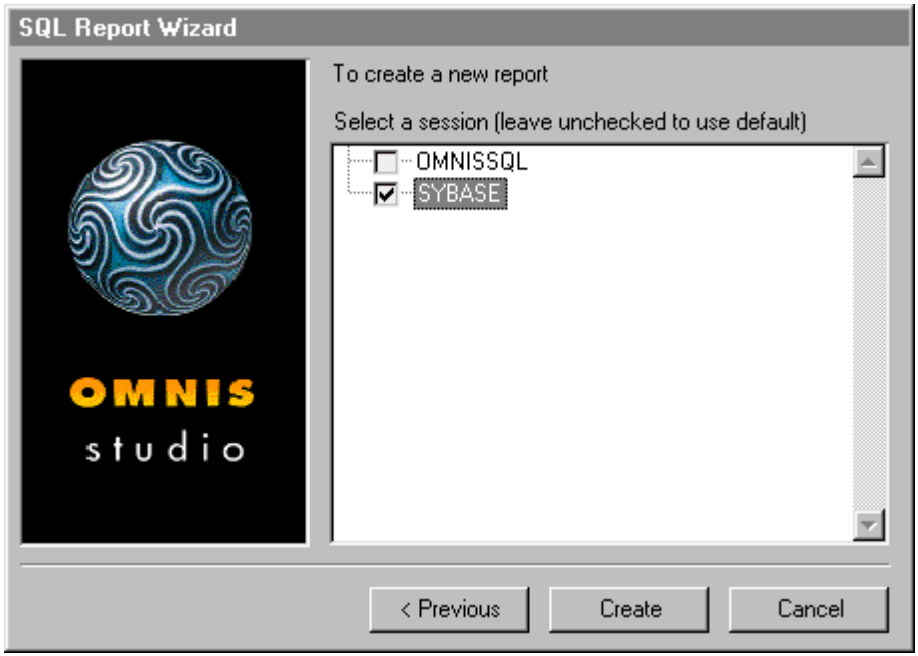
- Drag a SQL or OMNIS Report Wizard from the Component Store onto your library in the Browser
- Name the new report class and press Return, or click in the Browser



The Report Wizard displays all the available schema, query, or file classes in your library and lets you select which columns or fields you want to include on your report. You can expand a particular schema, query, or file to include or exclude individual columns or fields. If you check a class name without expanding it, *all* the columns or fields from that class are included on your report. The wizards *do not* let you include different columns or fields from different table or file classes, although a query class may contain columns from more than one schema class.

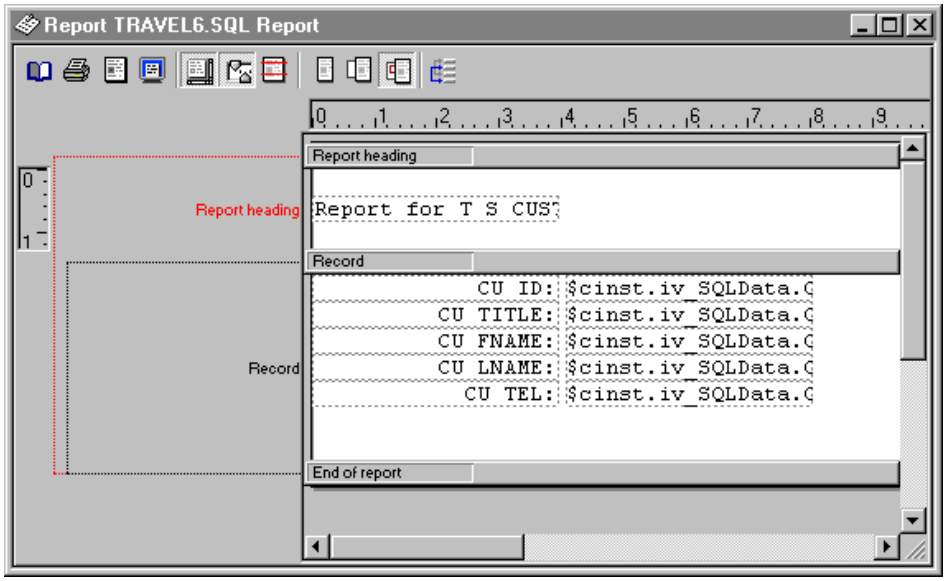
If you click Cancel at any time, the wizard is halted and the new class is removed from your library. To continue

- Choose a schema, query, or file class to base the report on, or select individual columns or fields in a class, and click on Next



For SQL reports only, the wizard prompts you to choose a session for the report, or you can leave the session names unchecked to use the default session.

- Finally, click on Create to create the report



When you finish in the Report Wizard the new report class is opened ready for you to modify or print. To modify your report you need to edit its properties. You can also add new report objects from the Component Store, and you can add methods to the report class or the objects on the report. These are all described in this and subsequent chapters.

To print your report

- Assuming the design report is on top and you are logged on to a server or OMNIS data file, you can click on the Print, Preview, or Screen Report button on the report design toolbar

or from the Browser you can

- Right-click on the new report class, and select Print Report from the context menu

Default Report Template or Wizard

The *default report template* or wizard is the one used when you create a new class using the Class>>New>>Report menu option in the Browser, and it's also the class shown under the Default Classes button in the Component Store toolbar. The default report template appears under the Report Classes button in the Component Store, shown by a check mark on the template icon. However, you can change the default template for report classes.

To change the default report template or wizard

- In the Component Store toolbar, click on the Report Classes button
- Right-click on the class or wizard you want to make the default
- Select the Make Default option from the context menu

For example, you can make the SQL Report Wizard the default template and from thereon, when you use Class>>New>>Report in the Browser, the SQL Report wizard will be called by default. When you click on the Default Classes button in the Component Store, the template or wizard you set as the default is shown. The remainder of this chapter assumes that the class called "New Report" is the default class.

Creating a New Report

This section describes how you can create a report class from the Component Store or from the Browser using the New Report template. A report you create in this way contains no fields, methods, or any other report objects, but it lets you create your report entirely from scratch.

To create a new report class

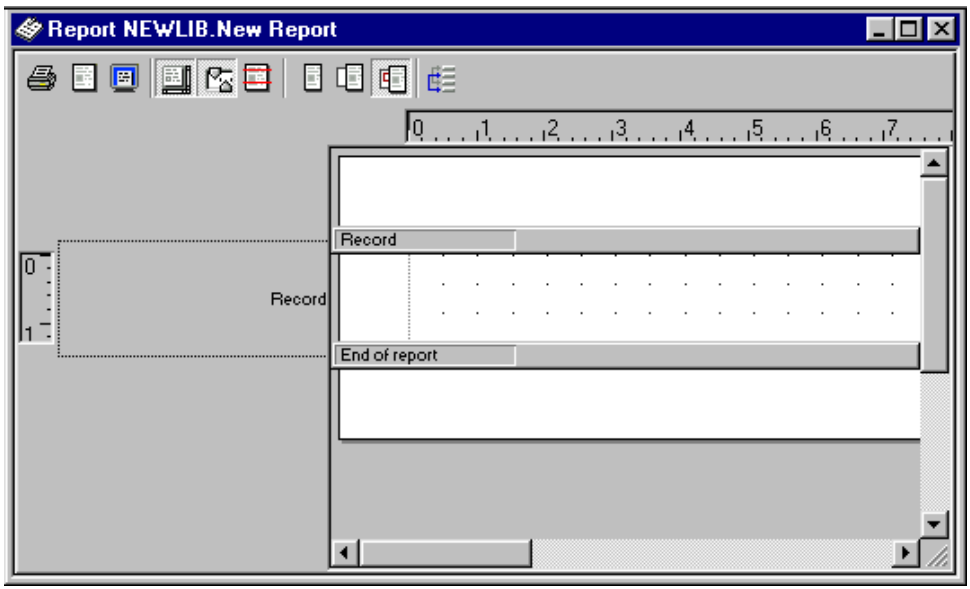
- Open your library in the Browser
- Display the classes in your library using the View>>Down One Level menu option on the Browser menu bar
- Drag the template called “New Report” from the Component Store onto the Browser

or

- From the Browser menu bar select Class>>New>>Report

The name of the new class is highlighted in the Browser, to continue

- Name the new report class
- Double-click on the report class to modify it

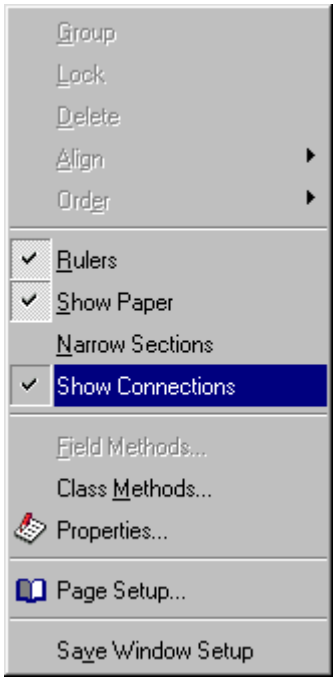


When you create a new window from the Component Store or Browser using the New Report template it does not contain any fields or report objects. The main white area in the report editor represents a page in which you construct your report. It contains two gray bars or *sections* in between which you place your data fields for the report. By default, there is a *Record* and *End of report* section. Any fields that you place in this section will be printed once for every record selected from your local database, or for every row selected from your server database. You can add other types of sections including Report and Page headers, Subtotal sections, Totals and Footer sections. These are described in *Report Sections* below.

Report Tools

The toolbar at the top of the report editor lets you set the page size, preview the report on screen, and show or hide connections between the different sections of the report as shown down the left-hand side of the report editor. In addition, you set the sort levels in your report from this toolbar. Position your mouse over each tool to see what it does.

Some of the options in the report editor toolbar are available using the report context menu by Right-clicking on the report background.



The Narrow Sections option displays the section markers as narrow lines which shows you how the report will look when you print it. The Class Methods option lets you add methods to the report class, and the Properties option shows the properties for the class. The Page Setup option or toolbar button opens the Page Setup dialog in which you can select the printer, and set the page size and orientation. This dialog will vary greatly across the different operating systems.

Report Properties

This section describes the properties of report classes and how you modify them using the Property manager to change the behavior of your report.

The properties under the General tab for a report class determine the behavior of the report while it is printing, and the visual aspects of the report structure. There are properties to set the margins for the report, the parameters for labels, and you can set up a design grid in your report to help you place objects accurately. The Sections properties let you show or hide the various section markers in the report class.

To view the properties of a report class

- Right-click on your report name in the Browser
- Select the Properties option in the context menu

or

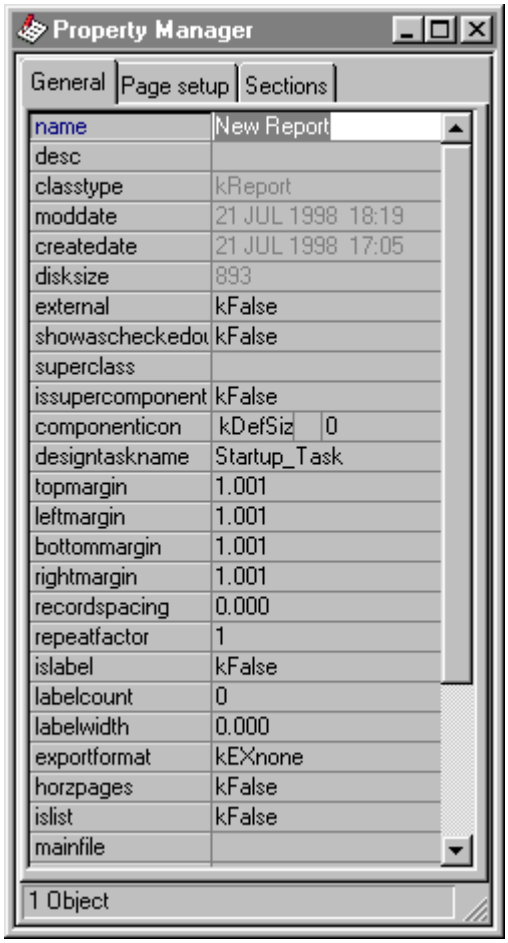
- Select your report name in the Browser and press F6/Cmnd-6

or, if you open a report to edit it and for some reason the Property Manager does not pop up, you can

- Click on the background of the report design screen (for report properties do not click on a field or section marker)

or

- Right-click on the background of the report design screen
- Select the Properties option in the context menu



Most of the properties are self-explanatory, including the standard class properties such as **name**, and **createdate**: this section describes some of the less obvious ones. The **recordspacing** property is the amount of space between each record or row on your report. The value of this property is used only if the **userecspacing** property for a section marker is enabled: by default the space between records is determined by the height of your record section.

The **repeatfactor** is the number of times each record is printed, for example, if you want two of each label set this property to 2.

The **islabel** property sets up the report as a label report, while **labelwidth** sets the distance between each record across the page and **labelcount** sets the number of labels across the page.

The **exportformat** property forces the report to export your data to one of the export formats; it is set to kEXnone by default. You can base a report on the data held in a list variable by setting the **islist** property to true and specifying the list name in **mainlist**. To print data from an OMNIS data file you must set the **mainfile** property to the main file class for the report.

The **shownames** property specifies whether report objects are displayed in the report editor with their object **name** or their **dataname** (the default). When you set **shownames** to kTrue, all objects on your report display their object names.

The units for all measurements in report classes depend on the setting of the OMNIS preference **usecms**. You can access the OMNIS Preferences from the Tools>>Options/Preferences menu option: set **usecms** to false for inches.

Page Setup Properties

Report classes have their own Page setup details stored as properties of the class.

pagesetupdata	stores the page setup data for the class; in the Property Manager you can click on the down arrow to open the page setup dialog. When the page setup is specified the property displays (Not empty); you can delete this text to clear the current page setup
orientation	the page orientation, the default is kOrientDefault
paper	the paper size or type, one of 50 or so constants, the default kPaDefault
paperlength	the length of the paper in cms or inches, the default is zero.
paperwidth	the width of the paper in cms or inches, the default is zero.
scale	the scaling factor in percent, the default is zero
copies	the number of copies, the default is zero

If any of the page setup properties remain empty or the default is used, the settings in **pagesetupdata** for either the class or the global **pagesetupdata** preference are used.

Report Field Types and Properties

This section describes the different types of report field or object available in OMNIS, and how you create them from the Component Store or from the Catalog. It also describes the properties of report fields, and how you modify these to change the appearance and behavior of fields on a report. Report sections are dealt with in *Report Sections* below.

You can place data fields, picture fields, graphs and section markers on your report, as well as background or graphic objects, including lines, ovals, rectangles, and text labels. See the *Window Classes* chapter for details about background objects.

The following objects are available for report classes in the Component Store.

Cursor
deselects other tools in the Component Store and reinstates the pointer tool

Entry
displays the data on your printed report; you need one report field for every variable or file class field you want to include on your report

Picture
field type that displays picture data, either Windows Metafiles, bitmaps or MacOS PICT files

Position
special type of section to structure and position parts of your report

Note that the Graph external component may also be available if you have it loaded. You can place a field on your report from the Component Store and set its **dataname** property in the Property Manager, or you can drag a variable or field from the Catalog to create a field automatically.

To create a report field from the Component Store

- Open your report class in design mode
- Drag the required field type from the Component Store onto your report between the first section marker and **End of report** sections

or to draw a field of a particular size

- Select the required field type in the Component Store by clicking on its icon
- Click and drag on your report, between the first section marker and **End of report** sections, to define the size of field you want

To create a report field from the Catalog

- Open your report in design mode
- Open the Catalog and locate your variable or file class field under the Variables tab
- Drag the required variable onto your report

When you place fields by dragging variables from the Catalog, OMNIS creates a field of the correct type and sets the **dataname** property to the name of the variable. For example, if you drag a Character or Number variable onto your report OMNIS creates a standard data field, or if you drag a picture variable OMNIS creates a picture field on your report. You cannot drag some types of variable, such as item references and binary variables.

There are a number of restrictions that apply to dragging variables from the Catalog to create fields.

– You can drag fields of type Date, Number, Sequence, Character, Boolean, and Picture only

– You cannot drag local and parameter variables

– You can only drop class and instance variables on to the class to which they belong

– You can only drop task variables onto classes belonging to the same design task

When you place an object on a report, using either of the above methods, the Property Manager opens showing the properties of the field, but if for some reason the Property Manager does not pop up you can open it in a number of ways.

To view the properties of a report object

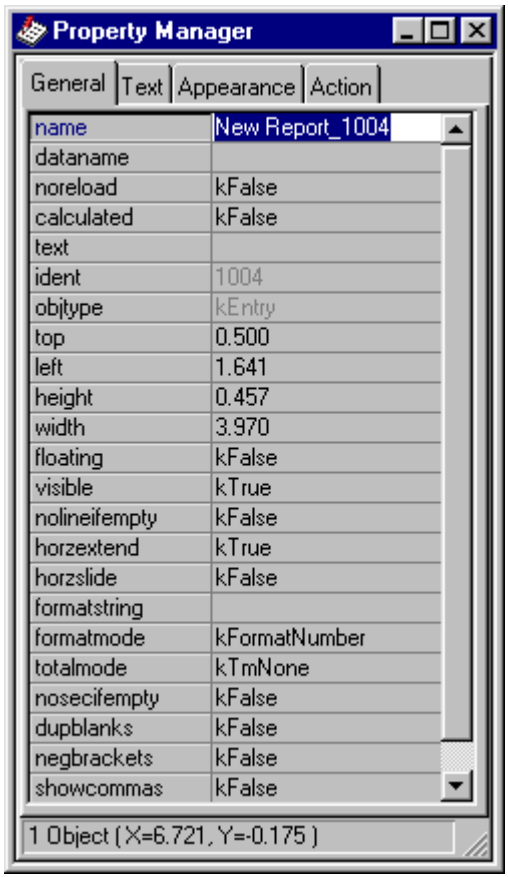
- Open your report in design mode
- Click inside the object, and the Property Manager opens automatically

or

- If the Property Manager is already open, click on the report object and press F6/Cmnd-6 to bring it to the top

or

- Right-click on the field
- Select the Properties option in the context menu



For report fields the Property Manager shows General, Text, Appearance, and Action properties. These properties control the behavior and appearance of the field on your report. The single-most important property for a report field is its **dataname**, that is, the name of the variable, column, or file class field the report field uses to display its data.

General Properties

All report fields have a **name** which can be any name you choose to identify the field, although it should be unique within the same report class. All fields must have a **dataname** to display data; this can be the name of a variable, column, or a file class field. You set the size and position of a field using the **left, top, width**, and **height** properties. The units for all measurements depend on the setting of the OMNIS preference **usecms**: set this to false for inches. Data and picture fields have the following general properties.

name	the name of the report field; this can be literally anything you choose to use to identify the object
dataname	the name of the variable, column, or file class field the report field uses to print its data
noreload	if true the value of the field is not reloaded at the start of totals or subtotals sections containing the field
calculated	if true the value of the field is calculated using the expression in the **text** property
text	the text for the object or the calculation for calculated fields
ident	the internal id of the object; note you cannot change this
objtype	the type of the report field; note you cannot change the type of existing report objects
top	position of the object relative to the top of the section containing the object, not the top of the report class or printed page
left	position of the object relative to the left-hand edge of the report, excluding the left margin
height	height of the object
width	width of the object
floating	if true, the field is floating, that is, you can position the object pixel-by-pixel; if false, the object snaps to the nearest row in the report
visible	if true the object is printed on the report, otherwise if false the field or its data is not printed; you may want to calculate a field or use its value elsewhere in the report, but hide it on the printed report
nolineifempty	if true and the field contains no data, the line containing the field will not print; this is useful for address labels when certain lines may be empty
horzextend	if true the field will expand horizontally to accommodate larger amounts of data
horzslide	if true and the preceding field has extended to accommodate more data, the current field will move to the right

formatstring	string containing formatting characters that formats the display of data on the report
formatmode	the type of formatting for the field: can be character, number, date or boolean
totalmode	totaling mode for the field, usually a field within a subtotals or totals section of a report: can be total, average, count, minimum, maximum
nosecifempty	if true and the field has no data, the section containing the field is not printed
dupblanks	if true and the field contains the same value as the previous row or record, the field remains blank
negbrackets	if true negative numeric values are shown in parentheses
showcommas	if true the thousand separator is displayed for number fields

Note that picture fields have the properties **name** down to **nolineifempty** only.

Normally report objects snap to the nearest row in the report class. However if you set an object's **floating** property to kFalse you can position the object precisely in your report.

When a report field appears in a totals or subtotals section, the value of the field is loaded automatically each time it is encountered in the totals section. However you can stop values being reloaded for an object by setting its **noreload** property to kTrue. When noreload is kTrue for an object, OMNIS does not reload its previous value when starting to print a section, nor does it restore it after printing the section. You can control field values in sections in your custom $print() methods. If the same variable is referenced by objects in more than one totals or subtotals section, you need to set noreload for each object. The noreload property is ignored for sort fields.

Text Properties

The text properties control font size, style, alignment, and color of the data in the field.

fieldstyle	the field style for the report object
font	font name for the field
fontsize	font size for the field
fontstyle	font style for the field
textcolor	text color of the field
align	alignment or justification of the field
fontextra	extra spacing for the text on the report

Appearance Properties

zeroempty if true zero values are displayed as blanks

shownulls if true the object displays “NULL” for undefined values

Picture fields also have the following property under the Appearances tab.

noscale if true pictures are not scaled to fit the size of the report field, otherwise pictures are scaled by default

Action Properties

autofind if true the field performs an automatic find (for indexed fields in an OMNIS data file only); when the field is triggered on the report, OMNIS looks for a exact matching record for the field

Background Objects

All data and picture fields on a report are referred to as *foreground objects*. All lines, rectangles, ovals, and any other graphic objects you place on your report are *background objects*. The latter do not hold data and cannot have methods attached to them or receive events; they are purely graphical devices for enhancing the appearance of your report. Any background objects you place in the Record section of your report will be printed for every record or row of data. Likewise any graphics you place in a subtotal heading section, the report header or footer section will be printed each time the section or page is printed.

You can create various types of background objects including text and labels from the Component Store.

To view background objects in the Component Store

- Click on the Background Components button on the Component Store toolbar

You place background objects on your report in exactly the same way as for fields. For example, you can click on a component icon in the Component Store and click-and-drag the mouse on your report to create an object of a particular size and shape.

Note that you can paste a picture or bitmap image onto your report as a background object using the **Edit>>Paste** or **Edit>>Paste from File** menu items. The former choice pastes a picture you have cut or copied to the clipboard; the latter pastes a Windows metafile or bitmap or a Mac PICT file directly onto the report background.

Report Sections

This section describes how you structure your report class using *report sections*. Sections are horizontal markers or dividers across the report class that structure and position the data when your report is printed. To create a complex report with headers, footers, subtotals, and totals, such as an invoice or catalog listing, you have to place the appropriate sections in your report class in the right order. When you enable the various sections in your report using the Property Manager, their position and order is handled for you automatically.

There are two sections that you must have in a report: the *Record* section indicates the start of the display of records or rows of data, and the *End of report* section indicates the end of the report. These sections appear automatically in every new report class.

The following section types are available.

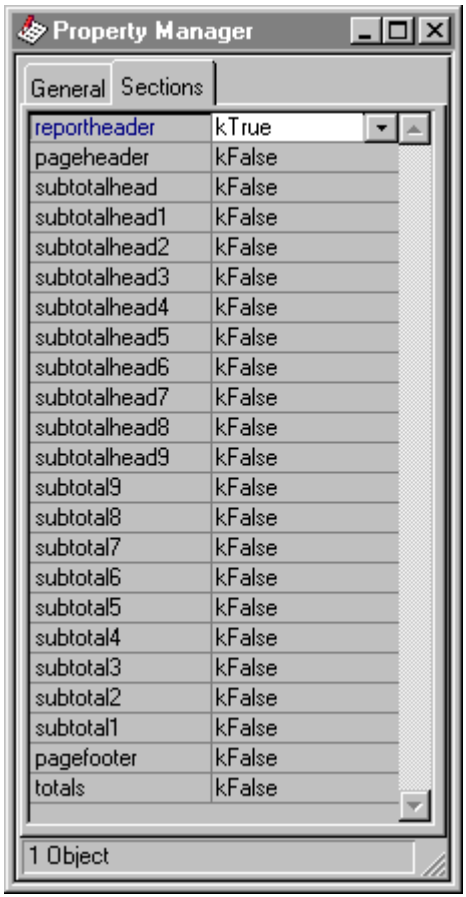
Report heading	defines the area at the start of the report, which prints only once; you can use this to create a report title page
Page header	defines an area at the top of each page below the top margin, printed at the top of each new page
Subtotal heading	prints before each subtotals section; it would normally contain column headings for your subtotal sections
Subtotal heading 1 to 9	each subtotal heading prints before its corresponding subtotal level
Record	defines the section containing the fields that print your data; the record section expands to accommodate your data which may extend over several pages when printed
Positioning	divides a section into two or more subsections; you can control exactly where on the page a positioning section is printed
Subtotals level 1 to 9	defines the fields that will print subtotals; you can have up to 9 levels of subtotaling
Totals	prints at the end of the report and defines the fields that you want to total
Page footer	defines the area at the bottom of each page; printed at the bottom of each new page of your report
End of report	defines the end of the report; must be present on every report

To enable a report section

- Open your report class in design mode
- Click on the background of your report to open the Property Manager

or, if the Property Manager does not open for some reason

- Right-click on the background of your report and select the Properties option from the context menu
- In the Property Manager, click on the Sections tab
- To enable a particular report section set the appropriate property to true

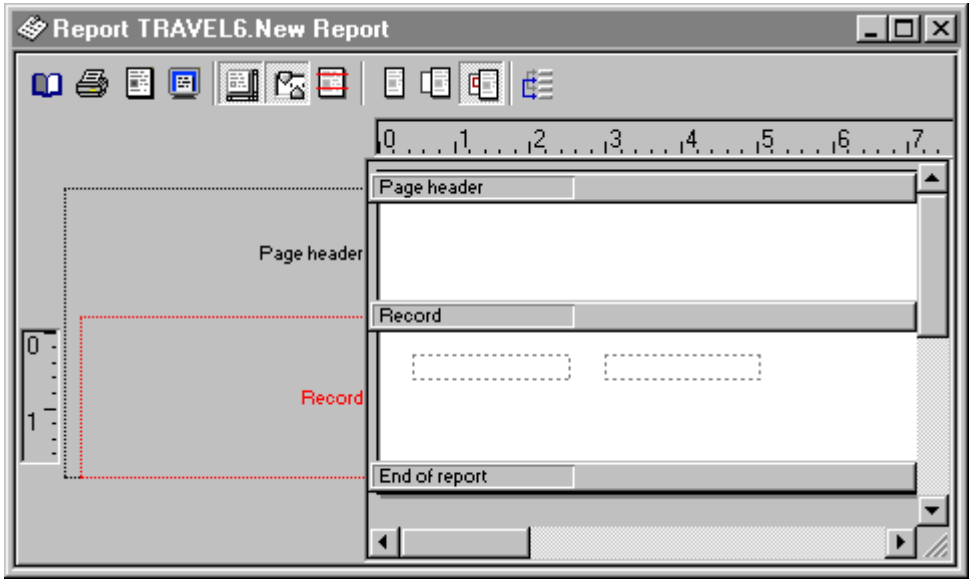


When you enable a particular report section it is shown on your report in the correct position. For example

To create a page header for your report

- View the properties for your report (as above)
- Set the **pageheader** property to true in the Property Manager

The Page header section will appear on your report above the Record section, or any subtotal headings if you have any.

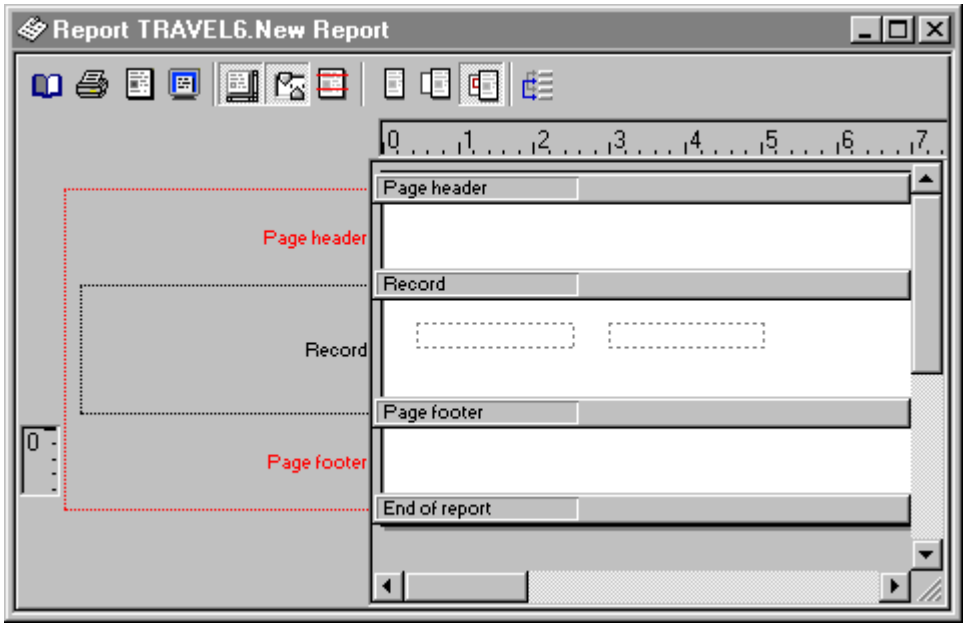


Any fields or graphics you place in the header section, that is between the Page header section and the next section marker will print at the top of each page. The following example will illustrate how the different sections or levels interrelate.

To create a page footer for your report

- View the properties for your report (as above)
- Set the **pagefooter** property to true in the Property Manager

The Page footer section will appear on your report below the Record section, or any Subtotals and Totals sections if you have any.



Any fields or graphics you place in the footer section, that is, below the Page footer section marker, will print at the bottom of each page. Note that the connection between the different sections is shown in the left margin of the report editor: the current section is shown in red.

When printing to a non-paged device such as File or HTML, by default the footer section is not printed. The Report header and first Page header sections are printed at the beginning of the report. However it is possible to force the footer section to be printed by calling $printsection(kFooter) for the report instance. The default positioning for a footer for a non-paged device is to follow on from where the last section stopped printing.

To change the height of any section, including the record, header and footer sections, you can click on the section marker (the gray bar) and drag it into position. All the sections below the one you move will adjust automatically.

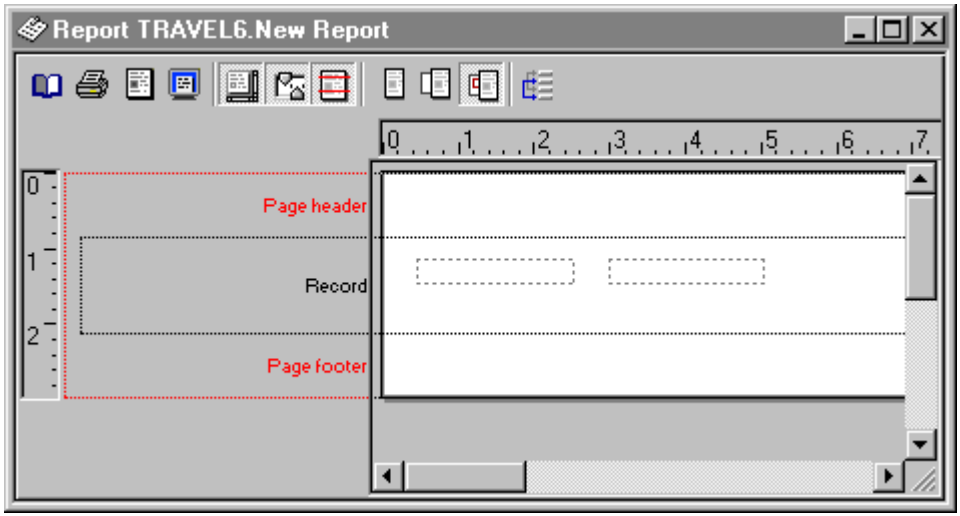
To show you more how the report will look when you print it, you can view the sections as narrow lines.

To view sections as narrow lines

- Click on the Narrow sections button in the report editor toolbar

or

- Right-click on the report background and select the Narrow Sections option



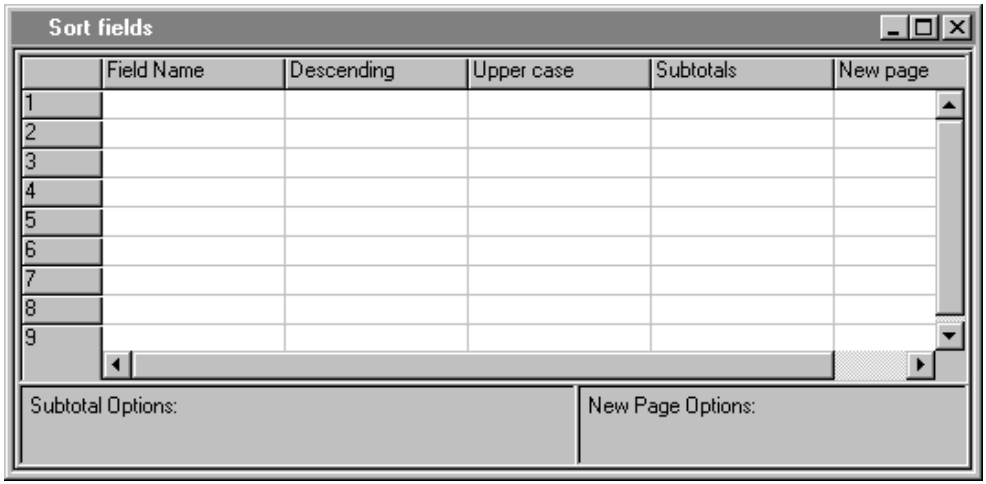
Sorting and Subtotaling

To implement sorting and subtotaling for your report, you need to specify the fields on your report to be sorted, and create subtotals sections containing those fields. *Sort fields* define how OMNIS subtotals the records or rows of data when printing a report. With no sort fields, OMNIS displays records in the order they are listed on your server, or in the order the data was inserted into your OMNIS data file. When you add sort fields to your report, the report will print subtotals when the values change in the sort fields.

To specify sort fields for your report

- Open your report in design mode
- Click on the Sort Fields button in the report editor toolbar

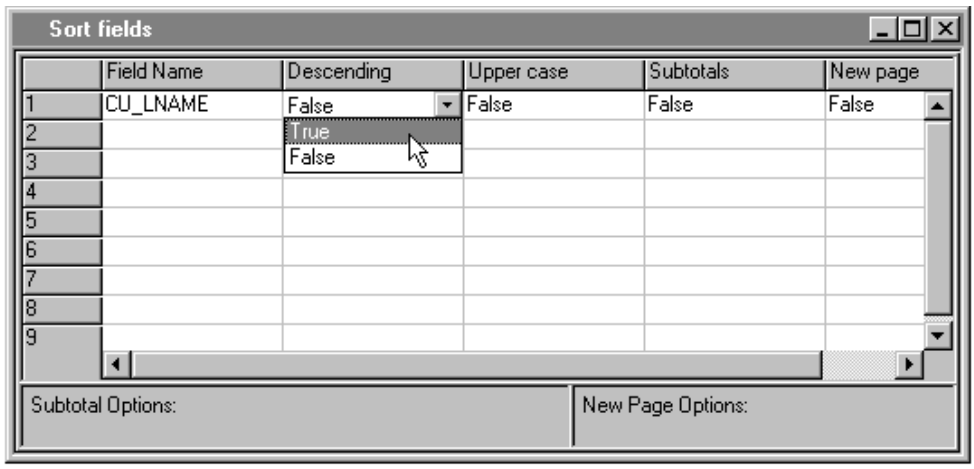
The Sort fields window opens.



In the Sort fields window you can specify up to nine sort fields by entering each field or variable name in the left-hand column. Note that you can use the Catalog to enter your field or variable names. These sort fields form a nested sequence of sorts on the records that trigger printing of up to nine nested subtotal sections.

To subtotal a field, place a copy of the field in the Subtotal section and select the appropriate **totalmode** for the field. This is independent of the sort fields, which trigger the printing of the Subtotal sections.

When you enter a field or variable name in the list of sort fields the sorting options are enabled for that field. You can enable any of these options by clicking on the cell and selecting true.



Each sort field has the following options.

– **Descending** sort
  sorts the field in descending (Z to A and 9 to 0) instead of the default ascending order

– **Upper case** conversion
  converts field values to upper case before sorting, so the use of mixed case in your database does not affect subtotaling or sorting

– **Subtotals** when field changes
  tells OMNIS to print subtotals using the corresponding subtotal section (1 to 9) when the value of the field changes; that is, if sort field 4 changes, subtotal level 4 will print

– **New page** when field changes
  starts a new page as well as printing a subtotal when the value of the field changes

When you enable the Subtotals or New page options for a sort field, you can specify the number of characters that must change before a subtotal is triggered or new page is printed.

Subtotal Sections

You can specify a *Subtotal heading* section in your report. It prints before the first Record section and successive Record sections following each Subtotals section. The subtotal heading can print column names and anything else you want to apply to each subtotaled Record section.

The *Subtotals* section prints whenever the Record section breaks on the corresponding sort field, with the subtotal printing before the record with the changed value. Since there are up to nine sort fields, you can have up to nine *Subtotal heading* and *Subtotal levels* numbered 1 through 9 corresponding to the sort fields specified in the report. The higher numbered sort fields are nested within the lower ones and hence change more often. That is, sort field 5 changes within sort field 4 which changes within sort field 3, and so on. Correspondingly, the Subtotal heading and Subtotals sections with higher numbers print more often as the sort fields change.

When you have multiple subtotals which print consecutively, the corresponding heading sections print one after another, starting with the one for the last subtotal. Subtotals and totals can be aggregations of several kinds, including sums, averages, counts, minimums, or maximums, depending on the field's **totalmode** property. OMNIS maintains the total for each subtotal printing, then resets the subtotal to zero for the next section.

The *Totals* section prints at the end of the report. As for subtotals, you place the fields to aggregate in this section, and OMNIS accumulates the aggregate values across the entire report. You can set the **totalmode** property for a field in the totals section.

Section Properties and Positioning

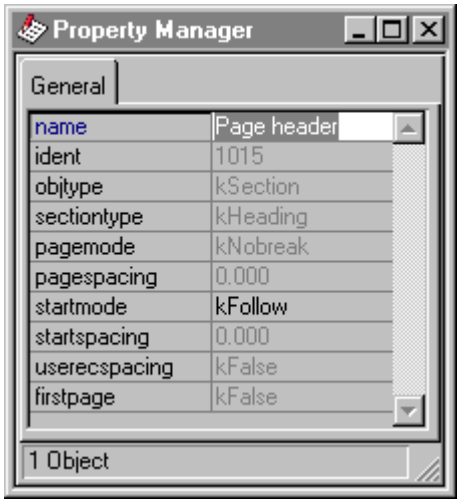
When you print a report, each section follows the previous section by default, and is positioned down the page according to the height of the previous section set in the report class. However for some types of section, you can control where a section prints and whether or not a new page is forced using the **pagemode** and **startmode** properties of the section. You can use a special type of section marker called a *position* section to print part of your report literally anywhere on the page. To do all these things you have to modify the properties of the appropriate section marker.

To view the properties of a section

- Open your report class in design mode
- Click on the appropriate section to view its properties in the Property Manager

or, if the Property Manager does not open or come to the top

- Right-click on the section marker and select the Properties option



Page Mode

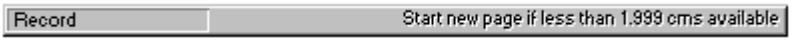
You can control whether or not *Record*, *Subtotals*, *Totals*, and *Subtotal heading* sections start a new page when they are encountered by setting their **pagemode** property. You can select one of the following options for this property.

– **Nopage**
does not force a new page; uses the pagination in the report class (the default)

– **Newpage**
  always starts a new page before this section

– **Testspace**
  starts a new page before starting this section if there is not the specified amount of space available on the current page

If you select the Testspace option, the **pagespacing** property is enabled in the Property Manager in which you can enter the amount of space required for the section. If this amount of space is not available on the page, a new page is started. The figure you enter in pagespacing is shown on the section marker.



OMNIS works with units that are 1/72 of an inch; therefore it may round exact numbers in centimeters or inches to the next real unit. For example, 1cm becomes 0.99cm.

Start Mode

All sections except for *Page footer* and *End of report* let you specify the **startmode**, which tells OMNIS where to start the section. You can choose one of the following options.

– **Follow** previous section
  starts the section on the line following the end of the previous section (the default)

– **Fromtop** of previous section
  starts the section n inches/cms from the *top* of the previous section

– **Fromend** of previous section
  starts the section n inches/cms from the *end* of the previous section

– **Fromtopmarg**
  starts the section n inches/cms from the *top margin* of the report

– **Frombottommarg**
  starts the section n inches/cms from the *bottom margin* of the report

When you choose one of the start modes the **startspacing** property is enabled in the Property Manager, which lets you enter a measurement for the startmode. The startmode and spacing is shown on the section marker.

OMNIS ignores previous section settings if the previous section was a *Page header* section or a *Positioning* section within a *Page header* section. The spacing comes before the page start test that examines the amount of space left on a page. OMNIS ignores top and bottom margin settings for reports that are not paged.

Note that when you set up a report to print labels, you can use the Fromtop or Frombottom options to set the spacing between your labels.

You can enter a negative value for the start spacing of a positioning section, for example Start –1.000cms from end of previous section. This allows you to align fields with the bottom of an extending field.

Record Spacing

The default spacing between records or rows of data on your printed report is determined by the height of the *Record* section in your report class. However you can override this spacing by setting the **userecspacing** property for the Record section. This property forces the report to use the vertical spacing set in the **recordspacing** property of the report class.

Positioning Sections

A *Position* section divides an existing section into two or more subsections, letting you reposition part of a section somewhere else on the printed page. For example, using a positioning section, you could divide a heading section into two parts to print the report title and description at the top of the page, and the report date at the bottom of the page.

A positioning section placed over the second line of a two-line extending field with the Follow previous section property prevents the second line from printing as a blank. You can also follow extending fields by a positioning section with Follow previous section to prevent them from writing over any fields below. A positioning section within a subtotal section lets you trigger a print position change by changing a sort field value.

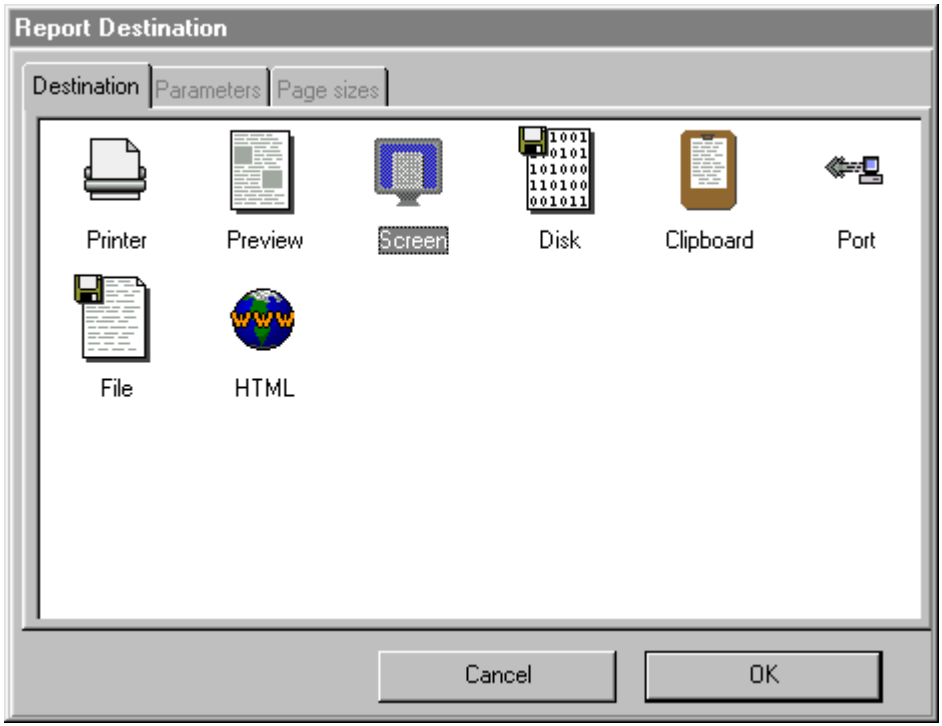
Printing Reports

OMNIS provides a wide range of choices for printing your reports, including sending your report to the printer, the screen, to a page preview, to a text or HTML file. Users can set the report destination using the File>>Print Destination menu option. In your finished library, you can provide a menu, popup menu, or toolbar button to print your report to the required destination. There are a number of commands that let you set the print destination, including *Send to screen, Send to file*, and *Send to clipboard*.

While you're creating your report class you may need to print it to try it out. You can use one of the buttons on the report editor toolbar to print the current report class. From these tools you can print to the current printer, a page preview window, or to the screen.

Report Destination Dialog

You select the output destination or device for your reports from the print destination dialog, available from the File>>Print Destination option on the main OMNIS menu bar. This dialog may also contain any custom devices, such as the HTML device.



The devices in the Print Destination dialog include

– **Printer**
the current printer

– **Preview**
reports are sent to a report preview window in OMNIS

– **Screen**
reports are sent to a report window in OMNIS; the default destination

– **Disk**
the report is sent to the file specified in the Parameters pane of the Print Destination dialog; the file is stored in a cross-platform proprietary binary format

– **Clipboard**
reports are sent to the clipboard in text format

- **Port**
  the report is sent to the port specified in the Parameters pane
- **File**
  the report is sent to a text file specified in the Parameters pane
- **HTML**
  the report is sent to an HTML file for display on your website
- **Memory** and **DDE/Publisher**
  are also available but by default are not visible in the Print Destination dialog (you can set their $visible device preference to make them visible)

To set the report destination

- Select File>>Print Destination from the main OMNIS menubar
- Select a report destination and click OK, or double-click on a destination

Printer

The Printer option sends the report to the current printer. Under Windows, selecting the printer as destination opens a list of installed printers, and changing to a new printer does not affect the default printer setup as defined in the Windows Control Panel. Note that you can change the page setup with the File>>Page Setup menu item.

Preview

The Preview option displays a full page on the screen. Text is “Greeked” if the screen size is too small, dots representing the characters so that the whole page fits the available screen area.

Screen

The Screen option is the default report destination at startup. It displays the report on the screen without the normal page margins. You can copy graphics in Windows metafile, bitmap or Macintosh PICT format from the screen and page preview report window by selecting an area with the mouse and using the Edit>>Copy menu item. You can copy text in the same way, and you can select and copy more of the report than is displayed on screen. When you drag-and-select data in your report, the window will scroll. The Select All option in the edit menu selects the whole report.

You can have more than one screen/page preview report open at a time. When you display reports on the screen, you can use the horizontal and vertical scroll bars to view the report, and the Page up, Page down, and arrow keys as well.

OMNIS displays a screen report window as soon as it has prepared the first page of data, that is, normally it does not wait for the report to finish. Therefore as you scroll a long report it may take a few seconds to print each page.

Disk

The Disk file report device sends the report output to a file on disk in a cross-platform binary format. If you double-click on the Disk icon in the Report Destination dialog OMNIS prompts you for a disk file name.

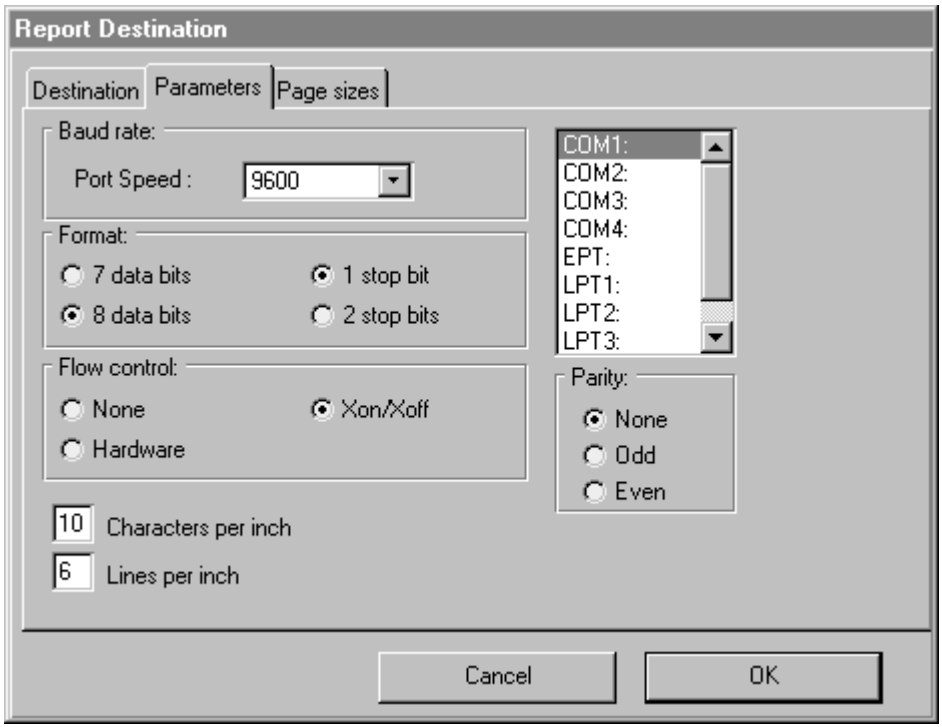
You can print to the Disk device on one platform, reload the file in OMNIS and print it on another platform. Alternatively, you can print the output from the Disk device using the File>>Print Report From Disk menu option, or using the *Print report from disk* command.

Clipboard

The Clipboard option sends the current report to the clipboard as an unpaged, text-only report suitable for pasting as text into other applications.

Port

The Port option sends the current report to a Windows serial or parallel port or a MacOS Modem or Printer port specified in the Parameters pane. This device also uses the settings in the Page sizes pane: see the File device.

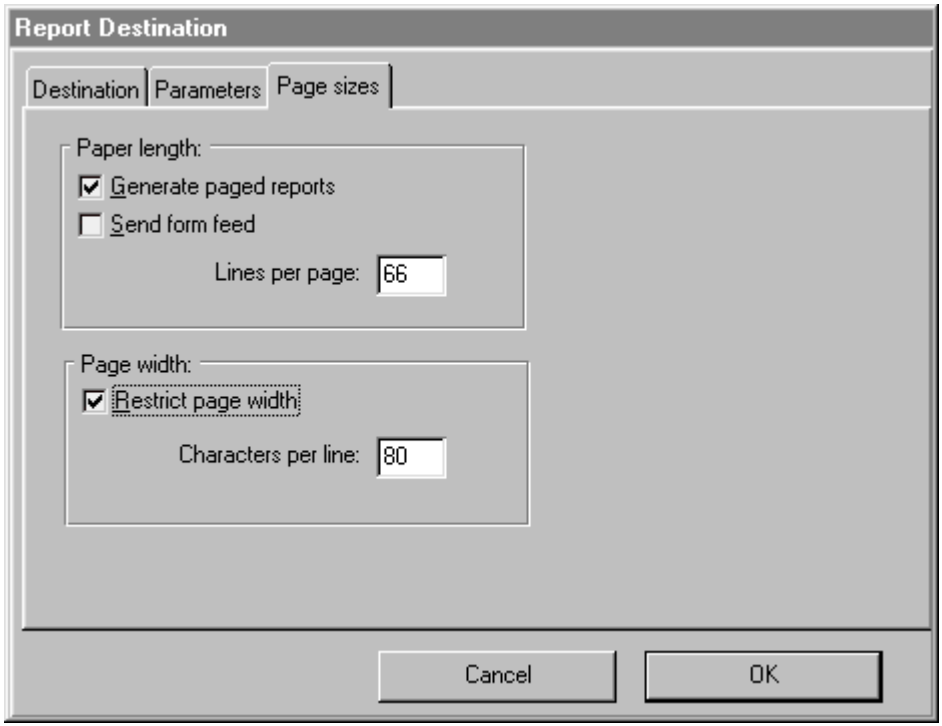


Only one program can have a particular port open; if a port is open in OMNIS and it is also, for example, the port used by the Spooler, then the Spooler will not be able to function.

Under MacOS, there is an option on the Parameters pane to Convert for Imagewriter. When selected, the characters beyond ASCII 127 convert to a combination of a character, backspace, and accent character so that the report can print accented characters and umlauts.

File

The File print destination sends the current report to a file. If you double-click on the File icon in the Report Destination dialog OMNIS prompts you for a file name. OMNIS does not close the file at the end of the report so you can append multiple reports into a single file. This option enables the Page sizes pane in the Report Destination dialog.



In the Page sizes pane you can specify the number of lines per page to use in reports printed to a file or a port. OMNIS stores the setting in the OMNIS configuration file.

If you check the Generate paged reports check box, you can also check the Send form feed check box, which tells OMNIS to terminate pages with a form feed, or fill in the Line per page field with a number of lines to which to pad out each page. Checking the Restrict page width option lets you enter the number of Characters per line.

HTML

The HTML report device is a custom device that prints a report to an HTML file on disk. When installed and loaded it appears in the print destination dialog and behaves like any

other standard printing device. You can send any OMNIS report to the HTML device and access and change the device using the notation. If you double-click on the HTML icon in the Report Destination dialog OMNIS prompts you for a file name.

The HTML printing device uses HTML tables and standard HTML tags to position and structure the output of the OMNIS report. The default background color of the HTML file is white. The color of the text in the original report class is retained in the HTML output file. Where possible, the device converts any image or picture data into JPEG images, which are written to disk and linked to the output HTML file.

Memory device

The Memory device lets you send a report to a binary variable or field, which you can hold in memory or save in a database. At a later date you can reload the contents of the binary variable or field and print the report to the printer or any other destination. You can access this device using the notation only (by default it is not shown in the Print Destination dialog). You can print the output from the Memory device using the *Print report from memory* command.

DDE/Publisher Device

The DDE/Publisher lets you send a report via DDE under Windows, or to an edition under MacOS. You can access this device using the notation only (by default it is not shown in the Print Destination dialog).

Report and Field Methods

You can create a report class, add fields and objects to the report from the Component Store, but to print sophisticated reports you will need to add some programming behind the fields and sections on your report. To do this, you need to write code that uses the OMNIS print commands or methods. You can add *class methods* to the report itself to control printing, and you can add *field methods* to each field or section marker on your report to control things like the interval breaks and subtotals.

You can add up to 501 methods to each field or section on your report, and a further 501 methods to your report class. You enter the methods for a report class and its fields using the method editor.

When you create a report from the Component Store it contains a $construct() and $destruct() method by default. You can add code to these methods to control the opening and closing of the report instance. For example, if your report uses a list you can build the list in the $construct() method in the report. You can use the $open() method or the *Prepare for print* command to open a report instance, you can finish a report using $endprint() or the *End print* command, and you can close a report instance using the $close() method. 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 record by record using the *Print record* command, or print an entire report using the *Print report* 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; there is no equivalent method for the *Print report* command. 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.

To add a method to a report class

- Open your report class
- Right-click on the report background to open the report context menu
- Select the Class Methods option
- Right-click in the method list in the method editor and add your method

To add a method to a report field or section

- Open your report class
- Right-click on the field or section to open its context menu
- Select the Field methods option
- Right-click in the method list in the method editor and add your method

The next section in this chapter describes the type of methods you can add to report classes and objects.

Report and Printing Notation

This section describes the properties and methods available for report classes, report objects, sections, report instances, and the printing devices available in OMNIS.

Print Devices and the Current Device

The following properties under $root handle the group of currently installed print devices or destinations, and the current printing device.

– $devices
group of currently installed printing devices, including Printer, Preview, Screen, Disk, Memory, Clipboard, Port, File, DDE/Publisher, and any custom devices you may have installed. You can set a reference to a device by using

```
Set reference MyRef to $devices.Screen
```

– $cdevice
the current printing device or report destination. You can change $cdevice by assigning a device from the $devices group, for example, to specify the screen as the current device use one of:

```
Calculate $cdevice as kDevScreen
Calculate $cdevice as $devices.Screen
Calculate $cdevice as $devices.$findident(kDevScreen)
```

Print Devices

The $root.$devices group contains the currently installed printing devices plus any custom devices you may have installed. A device has the following properties; $canassign for these properties is true unless specified.

– $name
the name of the device; $canassign is false

– $title
the string used to identify the device in the Print destination dialog

– $iconid
the id of the icon for the device as displayed in the Print destination dialog, zero by default which means the device uses the default icon

– $ident
a unique numeric identifier for the device in the $devices group; $canassign is false

– $visible
if true, the device is shown in the Print destination dialog

– $isopen
  returns true if the device is open and in use; $canassign is false

– $istextbased
  returns true if the device is text-based, otherwise, the device is image-based, such as Printer, Screen, or Preview; $canassign is false

– $cangeneratepages
  this is a read only property. If it returns true, the device can generate pages and all normal page headers and footers will be printed. If it returns false, only the report header and first page header are printed. No footer section is printed.

– $cankeepopen
  returns true if the device can be kept open for long periods of time, such as the File device; otherwise, the device should be opened prior to printing and closed immediately after printing has finished, for example you must do this for the Printer; $canassign is false

You can use the following methods for a device; $cando() returns true if a device supports the method.

– $open()
  opens the device ready for printing or transmitting text or data. Some devices such as the Screen or Preview can only be opened from a print job when printing a report

– $close()
  closes the device, if the device is open

The following example prints two reports in the same print job, and uses the $open() and $close() methods to initialize the Printer.

```
Set reference theDevice to $devices.Printer
If theDevice.$isopen         ;; check if printer is in use
   If theDevice.$canclose()
      Do theDevice.$close()
   Else
      Quit method kFalse    ;; if device can’t be closed
   End if
End If
Do theDevice.$open() Returns ok      ;; open the printer
If ok    ;; print the reports
   Set report name reportOne
   Print report
   Set report name reportTwo
   Print report
   Do theDevice.$close() Returns ok       ;; close the printer
End If
```

– $canclose()
returns true if the device can be closed. If you opened the device using $open(), this method returns true; if you opened the device via a print job and the job is still in progress, it returns false

– $sendtext(cText, bNewLine, bFormFeed)
sends the text in cText to the current device; all normal character conversion takes place. If bNewLine is true, the device advances to a new line or sends an end of line character; if bFormFeed is true, a new page is started, or a form feed character is sent. Data is sent in parameter order: first text, then the new line, then the form feed.

The following example sends some text to the File device.

```
Set reference theDevice to $devices.File
If theDevice.$sendtext.$cando()
   Do $prefs.$printfile.$assign('HD:MyFile')
   Do theDevice.$open() Returns ok
   If ok
      Do theDevice.$sendtext('Some text',kTrue) Returns ok
      Do theDevice.$sendtext('More text',kTrue) Returns ok
      Do theDevice.$close() Returns ok
   End If
End If
```

– $senddata(cData[,cData1]…)
sends the specified data in a binary format to the device; no character conversion takes place unless the data is of type kCharacter. If more than one parameter is specified the data is sent in individual packets

When using the $senddata() method you must consider type conversion. The method expects binary data, and therefore any data which is not in a binary format is converted to binary. For example, if you pass an integer variable, the data is converted to a 4 byte binary. In some cases, due to cross platform incompatibilities, if you want to be certain of the order in which the data is sent, and of the values which are sent, you should use variables of type Short integer (0 to 255), for example

```
Calculate myShortInt1 as 13
Calculate myShortInt2 as 10
Do myDevice.$senddata( myShortInt1, myShortInt2 )
```

You can send raw data to the Port or File device. The following example prints a report to a binary variable and sends the binary data to the Port device.

```
; print a report to a binary variable
Do $cdevice.$assign($devices.Memory)
Do $prefs.$reportdataname.$assign('myBinaryField')
Set report name myReport
Print report
; now send the report to the port
Set reference theDevice to $devices.Port
If theDevice.$senddata.$cando()
   Do theDevice.$open() Returns ok
   If ok
      Do theDevice.$sendata(myBinaryField) Returns ok
      Do theDevice.$close() Returns ok
   End If
End If
```

– $flush()
  flushes the device. For the File device $flush() will ensure all data is written to disk; you can safely call $flush() for devices which do not support this method; $cando() returns true for all devices that support $senddata() or $sendtext()

Global Printing Preferences

There are a number of OMNIS preferences under $root.$prefs that handle the print devices and their parameters. You can set these using the Property Manager or using the notation.

– $reportfile
  the full path and file name for the Disk device

– $printfile
  the full path and file name for the File device

– $editionfile
  the full path and file name for the DDE/Publisher device

– $pages
  the page or page numbers to be sent to the device; all devices support this property. You can specify pages as a comma-separated list or range of pages separated by a hyphen, or any combination. Prefixing a range with an "e" will print even pages within the range, or with an "o" will print odd pages within the range. For example 1,3,7,10-15,25-20,e30-40,o30-40

– $reportdataname
  the name of the binary field for the Memory device

– $reportfield
  the name of the window field for a Preview or Screen report; if you specify this property the report is redirected to the window field

– $windowprefs
the optional title and screen coordinates for a Screen or Preview window; the syntax is the same as the *Open window command*, such as My Title/50/50/400/300/STK/CEN; the title is also used as the document name when printing to the Printer

– $waitforuser
if true, method execution is halted until the user closes the Screen or Preview window

– $hideuntilcomplete
if true, a Screen or Preview window remains hidden until the report is finished

The following example specifies the Screen as the current device and sets up the preferences for the report window.

```
Do $cdevice.$assign(kDevScreen)
Do $prefs.$windowprefs.$assign('MyTitle/20/20/420/520/CEN')
Do $prefs.$waitforuser.$assign(kFalse)
Do $prefs.$hideuntilcomplete.$assign(kTrue)
```

– $charsperinch
the number of characters per inch when printing to a text-based device

– $linesperinch
the number of lines per inch when printing to a text-based device

– $generatepages
if true, reports generate paged output when printing to text-based devices, that is, page headers and footers are generated as normal; otherwise if false, only one report header and page header is printed at the beginning of the report

– $linesperpage
the number of lines per page when $generatepages is true

– $restrictpagewidth
if true, the width of a page is restricted when printing to text-based devices

– $charsperline
the number of characters per line when $restrictpagewidth is true

– $sendformfeed
if true, form feeds are sent to text-based devices after each page

– $appendfile
if true, data is appended to the current print file specified in $printfile, otherwise if false, the file is overwritten when printing to the File device; note if the device is already open prior to printing a report, the file is appended to regardless

– $istext
if true, forces a non-text device to behave like a text-based device using the same preferences as text-based devices

– $portname
the name of the port when printing to the Port device

– $portspeed
the port speed setting when printing to the Port device

– $porthandshake
the handshake when printing to the Port device; this can be kPortNoHandshake, kPortXonXoff, or kPortHardware

– $portparity
the parity checking when printing to the Port device; this can be kPortNoParity, kPortOddParity, or kPortEvenParity

– $portdatabits
the number of databits when printing to the Port device; this can be kPort7DataBits, or kPort8DataBits

– $portstopbits
the number of stop bits to be used when printing to the Port device. This can be kPort1StopBit or kPort2StopBits

The following example sets up the preferences for the Port device.

```
Do $prefs.$portspeed.$assign(9600)
Do $prefs.$porthandshake.$assign(kPortNoParity)
Do $prefs.$portdatabits.$assign(kPort8DataBits)
Do $prefs.$portstopbits.$assign(kPort1StopBit)
Do $prefs.$porthandshake.$assign(kPortXonXoff)
Do $prefs.$charsperinch.$assign(10)
Do $prefs.$linesperinch.$assign(6)
; Note $charsperinch and $linesperinch are used for
; all text-based devices
```

There is also a group of Page setup properties under $root.$prefs giving access to the global page settings. These are

– $orientation
the page orientation; this can be kOrientDefault, kOrientPortrait, or kOrientLandscape

– $paper
the paper type, a constant; one of 50 or so paper sizes or types including US Letter, European A sizes, envelope sizes, custom sizes, and so on

– $paperlength
the length of the paper in cms or inches depending on the $usecms preference

– $paperwidth
the width of the paper in cms or inches depending on the $usecms preference

– $scale
the scaling factor in percent

– $copies
the number of copies

Report Instances

Report instances have the following methods.

– $printrecord()
prints the Record section; same as the *Print record* command

– $printtotals(*section*)
triggers a subtotal or totals section; *section* is the highest level subtotal to be printed, a constant, such as kSubtotal5 or kTotals

– $printsection(*section*)
prints a report section; *section* is a constant, kRecord, kTotals, and so on, or a reference to a field on the report instance

– $accumulate(*section*)
accumulates the subtotals and totals section, and is sent during the printing of a record section

– $checkbreak()
checks if a subtotal break is required, returns a constant: kSubtotal1 to kSubtotal9 or kNone if subtotal break is not required

– $skipsection()
skips the current section; if you call this during $print() for a field, no further fields will be printed for that section

– $startpage(*pagenumber*)
starts a new page; adds the page header section to the page, and for the first page also adds the report header section

– $endpage(*pagenumber*)
ends a page and adds the footer section to the page; without parameter ends all pages which have been started

– $ejectpage(*pagenumber*)
ejects a page; without parameter ejects all pages which have been ended and not ejected; this method ejects pages which have an active section intercepting their boundary when the section has finished printing

– $endprint()
finishes the report; prints the final subtotals and totals sections and ejects all the remaining pages

– $openjobsetup()
  opens the job setup dialog. You can call this method immediately after $open() for a report; if it returns kFalse as the result, the user has selected Cancel, and you should close the report instance. You cannot call $openjobsetup() during $construct() since a print job is not created until $construct() finishes.

– $cdevice
  reference to the printing device for the instance; if you wish to change the device, you must do so before returning from $construct() of the report instance, and before you start printing the first record. For example, execute the following at the start of $construct() to specify the page Preview device for the current instance

```
Do $cinst.$cdevice.$assign($devices.Preview)
```

The device preferences are listed under the global printing preferences. Report instances have their own printing preferences which are local to the instance. They take their initial values from the global printing preferences. You can only assign values to these properties in $construct(), and before you start printing the first record.

The $firstpage property always returns 1, and $canassign() is false. The $lastpage property returns the last page. You cannot set $lastpage in the report instance to reduce the number of pages generated, that is, once pages have been generated they cannot be removed from a print job.

Note that the $pageheight property of a report instance returns the height of the printable area excluding the margins, headers, and footer areas of the report.

The following method prints a report from a list and uses a *For* loop to print the report record by record.

```
Do $reports.Report1.$open('*') Returns Myreport
For lineno from 1 to Mylist.$linecount step 1
   Do Mylist.[lineno].$loadcols()
   Do Myreport.$printrecord()
End For
Do Myreport.$endprint()
```

The following method generates subtotal breaks every fifth record. The $reccount property is incremented and the subtotals accumulated manually.

```
Do $reports.Report2.$open('*') Returns Myreport
For lineno from 1 to Mylist.$linecount step 1
   Do Mylist.[lineno].$loadcols()
   Calculate Myreport.$reccount as Myreport.$reccount+1
   Do Myreport.$printsection(kRecord)
   Do Myreport.$accumulate()
   If mod(lineno,5)=0
      Do Myreport.$printtotals(kSubtotal1)
   End If
End For
Do myreport.$endprint()
```

Page Setup Report instance properties

You can change the page setup information of a report instance without effecting the global settings. The properties which can be set are;

– $pagesetupdata
this can only be calculated prior to printing the first record; the best time is during $construct. When a print job has started, $canassign returns kFalse.

– $orientation, $paper, $paperlength, $paperwidth, $scale, and $copies
any of these properties can be changed at any time during a print job, and will effect the next page to be generated. When $startpage for a page has been called, changing these properties will take effect from the next page onwards. A good time to make changes for the next page is during a $endpage for the current page, but it can be done from anywhere prior to the $startpage call for a page to be effected

Once a print job is complete and $endprint has been called, $canassign returns kFalse for all these properties.

Report Field and Section Methods

Report fields and sections contain a $print() method that controls that particular field or section when it is printed. Every time a field or section is encountered during printing its $print() method is called, so for fields in the report Record section $print() is called for every row of data. You must end your own custom $print() methods with a *Do default* command to carry out the default processing for that line after your code has executed.

For example, the following $print() method for a report field prints the field in bold if its value is greater that 1000.

```
If parm_value>1000
   Do $crecipient.$fontstyle.$assign(kBold)
Else
   Do $crecipient.$fontstyle.$assign(kPlain)
End If
Do default
```

Report Object Positioning

When a report field prints, its position and data are passed to its $print() method; when a report section prints its position only is passed. You can set up parameter variables of type Field reference in the $print() method for a report section or field to receive its position and data. You can manipulate the position variable using the report object positioning notation. If you change the position of a section all objects in that section are affected together with all subsequent sections in the report. Making changes to the position of an object does not affect other objects.

A report position variable has the following properties.

– $inst
  the report instance to which the position belongs

– $posmode
  the mode of the report position, which is one of the following constants; assigning $posmode does not change the physical position of the object, but it does change its coordinates to the new coordinate system.

kPosGlobal	the position is global to the print job, relative to the top-left of the local area of the first page
kPosPaper	the position is relative to the top-left of the paper edge of the page specified by $posvertpage and $poshorzpage
kPosPrintable	the position is relative to the top-left of the printable area of the page specified by $posvertpage and $poshorzpage
kPosLocal	the position is relative to the top-left of the local area (excluding the header and footer sections, and the margins) of the page specified by $posvertpage and $poshorzpage
kPosHeader	the position is relative to the top-left of the header area (union of report and page header sections) of the page specified by $posvertpage and $poshorzpage
kPosFooter	the position is relative to the top-left of the footer area of the page specified by $posvertpage and $poshorzpage
kPosSection	the position is relative to the top-left of the section specified by $possectident

In addition, you can set $posmode to one of the following values to return the coordinates of an area on the page specified by $posvertpage and $poshorzpage.

kBndsGlobal	returns kPosGlobal coordinates. The top, left, width, and height are calculated to global coordinates of the local area of the page
kBndsPaper	returns kPosPaper coordinates. The top and left are zero, and the height and width are calculated to the height and width of the paper of the page
kBndsPrintable	returns kPosPrintable coordinates. The top and left are zero, and the height and width are calculated to the height and width of the printable area of the page
kBndsLocal	returns kPosLocal coordinates. The top and left are zero, and the height and width are calculated to the height and width of the local area of the page
kBndsHeader	returns kPosHeader coordinates. The top and left are zero, and the height and width are calculated to the height and width of the header area of the page
kBndsFooter	returns kPosFooter coordinates. The top and left are zero, and the height and width are calculated to the height and width of the footer area of the page

– $possectident
the $ident of the section when $posmode is kPosSection

– $posvertpage
the vertical page number when $posmode is *not* kPosGlobal or kPosSection

– $poshorzpage
the horizontal page number when $posmode is *not* kPosGlobal or kPosSection. This will usually be set to 1. Horizontal page numbers apply when horizontal pages are enabled

– $top
the top of the position in cms or inches local to its $posmode

– $left
the left of the position in cms or inches local to its $posmode

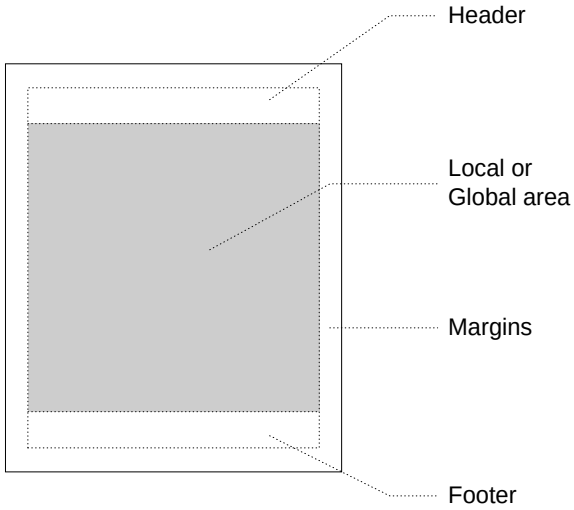
– $height
the height of the position in cms or inches

– $width
the width of the position in cms or inches

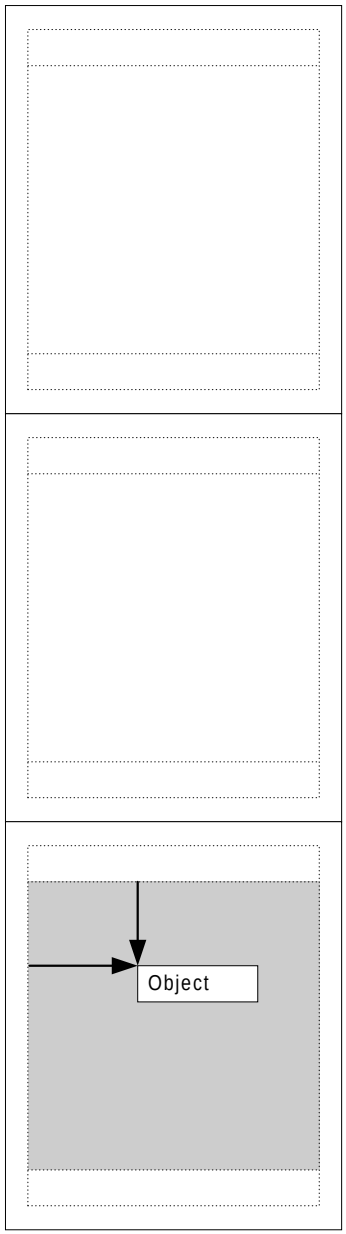
Measurements are in either cms or inches depending on the setting of the **usecms** OMNIS preference which you can change in the Property Manager using the Tools>>Options/Preferences menu option.

Page layout

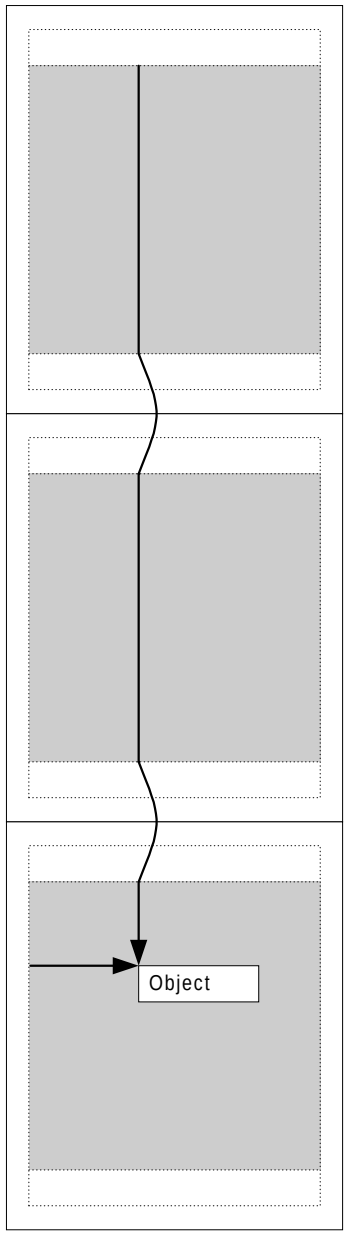
To understand the positioning notation it helps to look at the layout of the report on paper or screen. The area available for printing is limited to the printable area on the paper as determined by the printer or device. Within this space OMNIS reports print to the header, footer, and local or global areas, that is, the space remaining after subtracting the header, footer, and margins specified in the class. Note that OMNIS subtracts the margins specified in the class from the paper edge, rather than the boundary of the printable area.



The position of a report object, either a section or report field, is relative to the local area on the current page, or the global area for the entire report.



Local coordinates are relative to the local area on the current page



Global coordinates are relative to the global area for the entire report

The following example method produces a report with multiple columns by configuring itself according to the current paper size and orientation. The report class contains various

instance variables including iCurColumn, iMaxColumns, iLeftAdjust to handle columns and a global left adjustment. The data is taken from a list, but your data can be from any source. The Record section contains one field that gets its data from the list. Note the code for this method does the positioning and the *Do default* command prints the section.

```
; $print() method for Record section in a column report
; Declare Parameter var pThePos of type Field reference
; and Local var posBnds of Row type
; pThePos is in global coordinates and does not contain the page
; number, so make a copy and convert it to page-based coordinates
Calculate posBnds as pThePos
Calculate posBnds.$posmode as kPosLocal
; Fetch the global boundaries for the page; we can do this now since
; setting $posmode to kPosLocal set $poshorzpage and $posvertpage
Calculate posBnds.$posmode as kBndsGlobal
; Check if the bottom of the section will fit on the page
If (pThePos.$top+pThePos.$height)>(posBnds.$top+posBnds.$height)
   ; if it doesn't fit is there room for a new col on current page
   If (iCurColumn<iMaxColumns)
      Calculate iCurColumn as iCurColumn+1
   Else
      ; put the section at the top of the next page for column one
      Do posBnds.$offset(0,posBnds.$height)
      Calculate iCurColumn as 1
   End If
   ; now calculate the section’s top position based on posBnds.$top
   Calculate pThePos.$top as posBnds.$top
   ; calculate the section’s left pos based on current column number
   Calculate pThePos.$left as
             (iCurColumn-1)*$cinst.$labelwidth+iLeftAdjust
Else If not(pThePos.$left)
   Calculate pThePos.$left as iLeftAdjust
End If
Do default      ;; this prints the Record section
```

Printing Errors

The print manager reports the following error codes and text. You can setup error handlers to manage these errors.

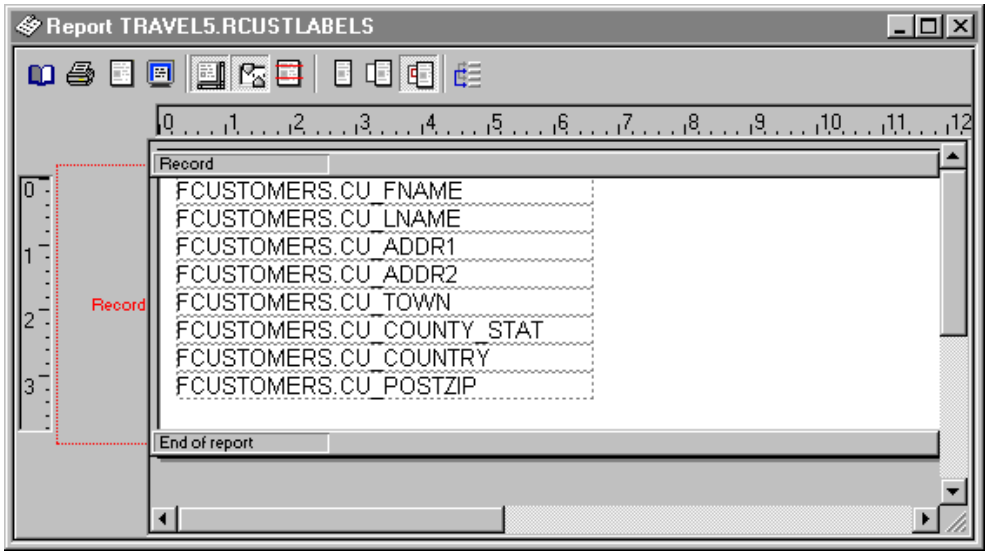
1001650	Non fatal print manager error
1001670	Fatal print manager error
1001680	Print manager system error; the code is shown in the error text
1001681	Other OMNIS error reported by print manager

Labels

To print labels in OMNIS you need to create a report class and set up its properties for label printing. You can create the report class using the standard SQL or OMNIS report wizards, or you can create an entirely New Report and add the fields yourself. This section uses the OMNIS Report wizard as the basis for a Customer address label, but the process is the same for any label report. Alternatively, you can use the ad hoc report tool to create a label report, described later in this chapter.

To create the basis of your label report

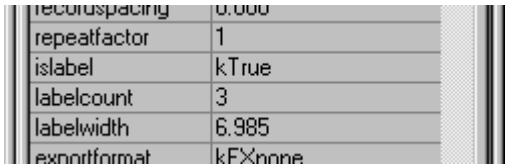
- Create a new report class using the OMNIS Report wizard and include the fields you want in your label
- Open the report class to modify it
- Delete the header section and any labels the report wizard places on the report class, but leave the data fields; your report class should look something like the following



To change this report class into a label report you need to change some of its properties, set the properties of the Record section to position your labels on the printed page, and as a further enhancement you can set the properties of some of the fields on the report to exclude empty lines. Note that all measurements use the current units set in the **usecms** OMNIS preference.

To set the label properties of a report class

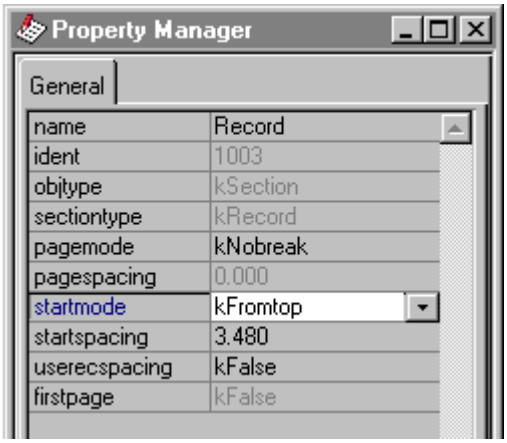
- Click on the background of your report class to view its properties



- Set the **islabel** property to kTrue
- Set the **labelcount** property to the number of labels across the page; for example, for standard 3 x 8 laser labels you set **labelcount** to 3
- Specify the width of a single label in the **labelwidth** property; if there are spaces between your labels, include the space in the label width, that is, the labelwidth is the distance between one record and next across the sheet of labels
- If you want to print more than one label for each row or record of data set the **repeatfactor** property, otherwise leave it set to 1 for a single copy of each label

To specify the distance between each row of labels down the page, you change the properties of the Record section in your report class.

- Click on the Record section to view its properties



- Set the **startmode** property to kFromTop, and in the **startspacing** property enter the distance between the top of one row of labels and the next going down the label sheet

Excluding Empty Lines

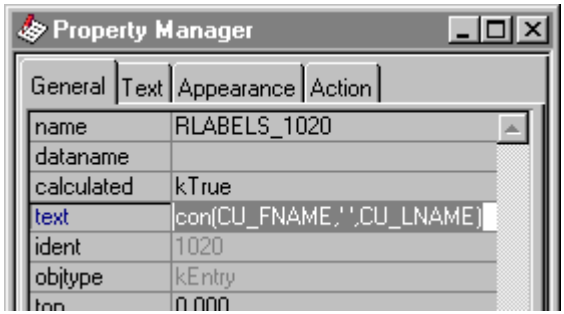
When you print your labels some of the fields may be empty and a blank line is printed. However you can stop a field from printing and move up all subsequent lines by setting the **nolineifempty** property for the field. For example, if your label includes two lines for the address you can set the **nolineifempty** property to kTrue for the second address field. In this case if the second address line is empty for a particular record, the line is not printed and subsequent fields move up one line. If any address field on your label is likely to be empty you ought to set its **nolineifmpty** property.

Using a Calculated Field

Rather than putting two separate fields on your label report for the Firstname and Lastname data, you can use a single calculated field and the *con()* function.

To create a calculated field

- Create a field on your report and view its properties

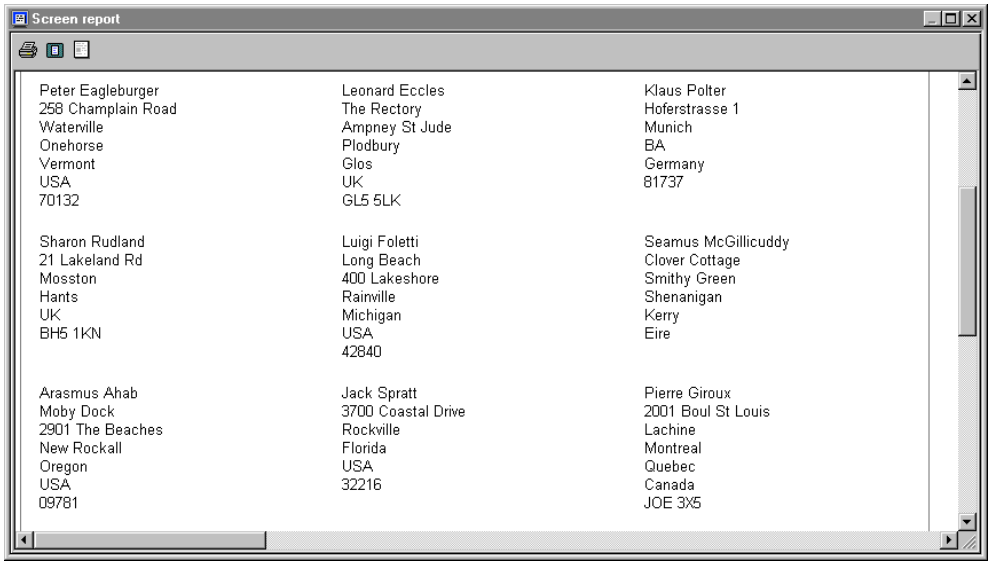


- Leave the **dataname** property empty, and set the **calculated** property to kTrue
- Enter the calculation in the **text** property, something like the following

```
con(CU_FNAME,' ',CU_LNAME)   ;; note space char is in quotes
```

The *con()* function concatenates the current values in the CU_FNAME and CU_LNAME fields and separates them with a single space character.

Using all the features described in this section, your label report should look something like the following when printed to the screen.



HTML Report Device

The OMNIS Studio Print Manager API has been made public, allowing you to create your own custom printing devices as external components and place them in the XCOMP folder. You can show your own custom printing devices in the Print Destination dialog, and use the printing preferences and notation to control your own devices. The HTML printing device is an external component and shows what you can do with custom devices. You can use the HTML report device in exactly the same way as the standard report destinations; there is no difference between internal and external output devices.

When the HTML external component is loaded in OMNIS, it registers an external output device with the OMNIS Studio Print Manager and shows the HTML icon in the Report Destination dialog. To print to the HTML output device you can set it up via the Report Destination dialog, or access it via the notation using the print device methods.

You can specify the HTML device using the notation as follows.

```
Calculate $cdevice as kDevHtml
Calculate $cdevice as $devices.Html
```

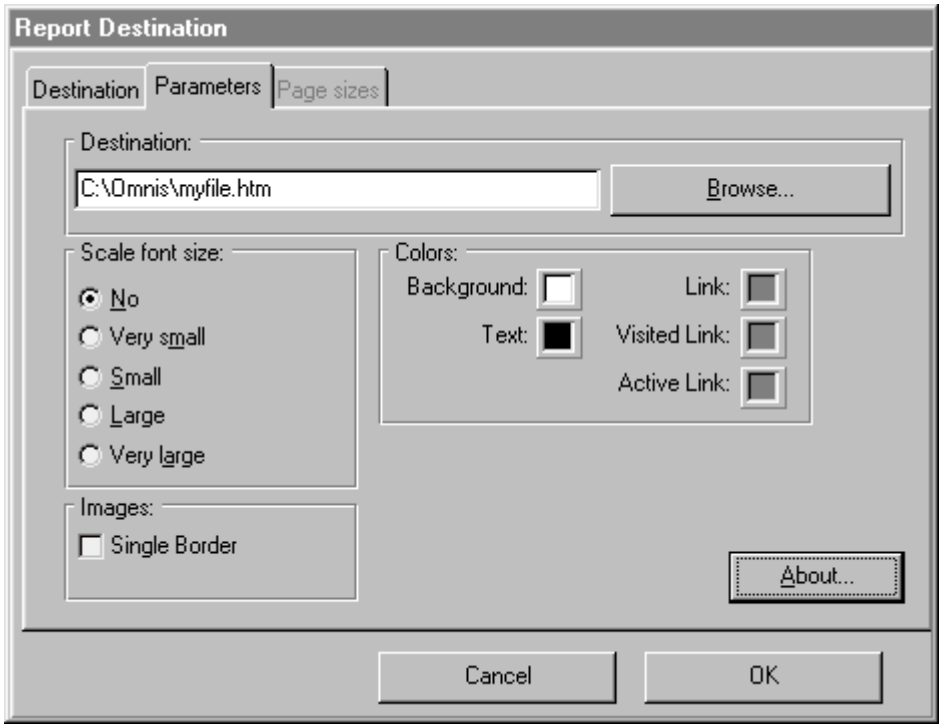
You can also set an item reference to the HTML device:

```
Set reference myDevice to $devices.Html
```

The constant kDevHtml is supplied by the HTML component at registration together with some other constants.

Setting the HTML Device Parameters

The HTML output device has several parameters which affect the overall appearance of the HTML document generated by the device. You can change some of these parameters in the Report Destination dialog and the notation, while some can be manipulated by the notation only.



The HTML device parameters are represented by constants which you can use in the notation. Some of them correspond to parameters in the Report Destination dialog.

Constant	Description
kDevHtmlFileName	the pathname of the destination HTML file
kDevHtmlFont1	largest point size which maps to HTML font size 1
kDevHtmlFont2	largest point size which maps to HTML font size 2
kDevHtmlFont3	largest point size which maps to HTML font size 3
kDevHtmlFont4	largest point size which maps to HTML font size 4

Constant	Description
kDevHtmlFont5	largest point size which maps to HTML font size 5
kDevHtmlFont6	largest point size which maps to HTML font size 6
kDevHtmlFont7	largest point size which maps to HTML font size 7
kDevHtmlImageBorder	whether JPEG images have a single pixel border
kDevHtmlUseRects	whether background rectangles are to be used to determine the background color of the HTML table cell which intersects the background rectangle
kDevHtmlBackcolor	background color of the HTML document
kDevHtmlTextcolor	default text color; any black text received from the print manager will be changed to this color
kDevHtmlLinkcolor	color for HTML text or pictures which are HTML links
kDevHtmlVLinkColor	color for links which have been visited
kDevHtmlALinkColor	color for links which are currently active
kDevHtmlTemplate	full path and file name of a template HTML file; it must already contain the basic framework for an HTML file, that is <HTML> <BODY bgcolor= …etc…> </BODY> </HTML>
kDevHtmlTemplateChars	the place holder contained within the template file which marks the point at which the report output will be inserted into the template, that is, “$$$$” the template file must contain this text, that is <HTML> <BODY bgcolor= …etc…> $$$$ </BODY> </HTML>
kDevHtmlScaleFont…	an additional single font scale factor; the following constants are available kDevHtmScaleFontNone: no scaling kDevHtmScaleFontVSmall: reduce HTML size by 2 kDevHtmScaleFontSmall: reduce HTML size by 1 kDevHtmScaleFontLarge: increase HTML size by 1 kDevHtmScaleFontVLarge: increase HTML size by 2

You can get and set the value of the device parameters using the following methods.

– $getparam(param constant)
  returns the value of the specified parameter

– $setparam(param constant, value [,param constant, value, …])
  sets the value(s) of the specified parameter(s)

For example

```
Do $devices.Html.$setparam(KDevHtmlFont1,6,KDevHtmlFont2,8)
Do $devices.Html.$getparam(kDevHtmlFileName) Returns MyPath
```

The value of all device parameters is stored in the OMNIS configuration file.

Sending Text or Data

It is possible to send text or data to some internal and external devices. The HTML device supports both. You can use the methods $sendtext() and $senddata() to send text and data, respectively.

When sending text, the HTML output device surrounds the given text with the correct HTML syntax, that is, it places begin paragraph and end paragraph statements around the text. You can send text with more than one call to $sendtext(), but still have the text appear in one paragraph. To do this, specify kFalse for the line feed parameter of the $sendtext() method. The device buffers the text separately, before adding a single paragraph to the document when you call $sendtext() with the line feed parameter set to kTrue.

When sending data, the device writes the data directly to the current position in the HTML file without any modification.

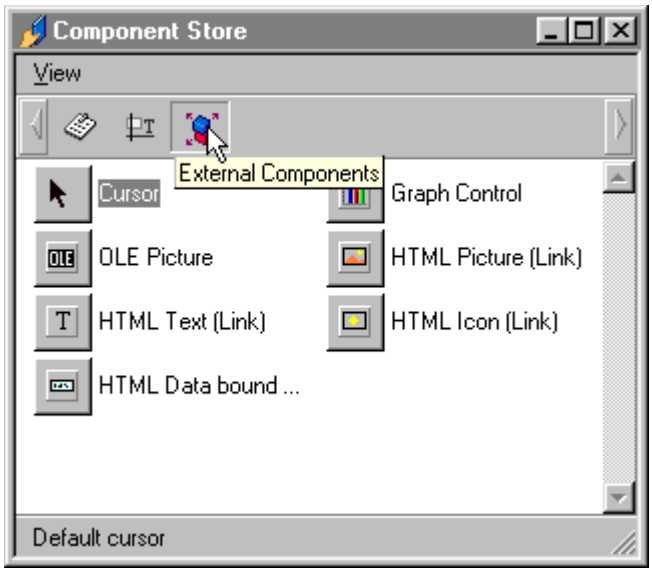
You can send text or data between reports, but not during printing, that is, while a report is being printed calls to $sendtext() and $senddata() are ignored.

The following method uses $senddata() to send data to an HTML file.

```
Set reference myDevice to $devices.Html
Calculate $cdevice as myDevice
Do myDevice.$setparam(kDevHtmlFileName,"C:\OMNIS\REPORT.HTM")
Do myDevice.$open() Returns ok
If ok
   Do myDevice.$senddata(myData1)
   Set report name Report1
   Print report
   Do myDevice.$senddata(myData2)
   Set report name Report2
   Print report
   Do myDevice.$senddata(myData3)
   Do myDevice.$close()
End If
```

HTML Report Objects

HTML report objects are special report objects that you can use to insert objects, such as other HTML documents, pictures, DLLs, or web site addresses, into your HTML reports. The HTML report objects are part of the HTML printing device and, when the report editor is the top window, they appear in the Component Store under the External Components button.



– HTML Picture
  standard picture object
– HTML Icon
  picture object which uses an icon id for its data
– HTML Text
  simple text object, like the standard label object
– HTML Data bound Text
  data bound text object, like a standard field; the data comes from an OMNIS variable or field

The HTML report objects have an $address property which you can set to the address or location of an HTML document, picture, DLL, or web site, for example, “results/result1.htm” or “http://www.omnis-software.com/”. The text can contain square bracket calculations, such as “www.[lvWebName]”.

If the HTML objects are printed to any other device other than the HTML device, they behave like their equivalent OMNIS field types, a standard picture or text field.

Ad hoc Reports

The ad hoc report plug-in offers you a quick way to generate reports and query your data. You can create ad hoc reports that access your server or OMNIS database. You can also use SQL reports with the OMNIS SQL DAM to access an OMNIS data file. The term *column* is used in this section to mean both server table columns and OMNIS file class fields.

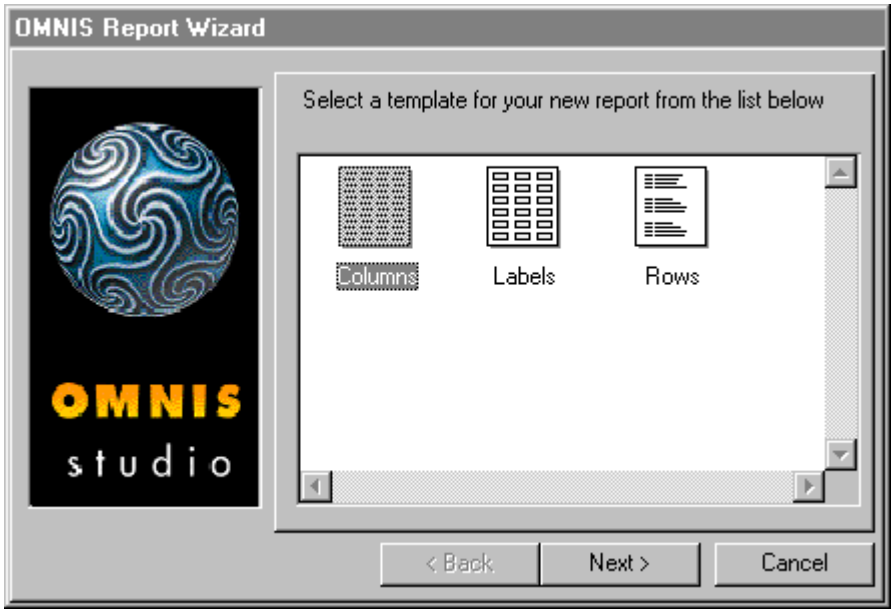
Ad hoc reports are based on standard OMNIS report classes, hence modifying an ad hoc report is very similar to modifying a standard report, which is described earlier in this chapter.

Creating Ad hoc Reports

You access the ad hoc report tool from the Reports menu. You need to install this menu to create or modify ad hoc reports. You also need to open a session using the SQL Browser, or an OMNIS data file using the Data File Browser.

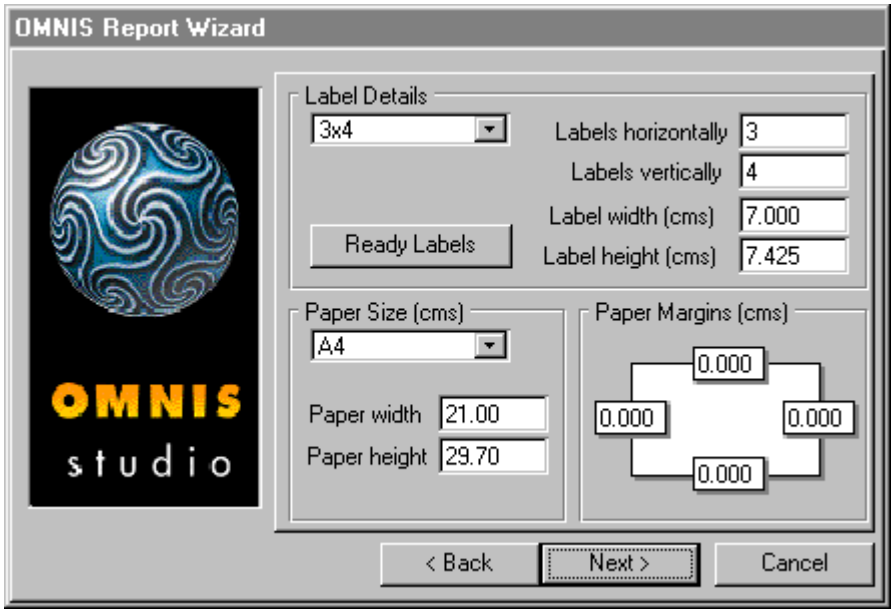
To create an ad hoc report

- Open the Tools menu on the main OMNIS menubar and select the Adhoc Reports option; the Reports menu is installed on the main menubar
- Open the Reports menu and select New



- Select a template and click on Next

If you select the Labels template the next pane in the wizard prompts you to enter details about your labels. You can set the paper size, label format, margins, and so on, or you can select a label format from the list of Ready Labels.

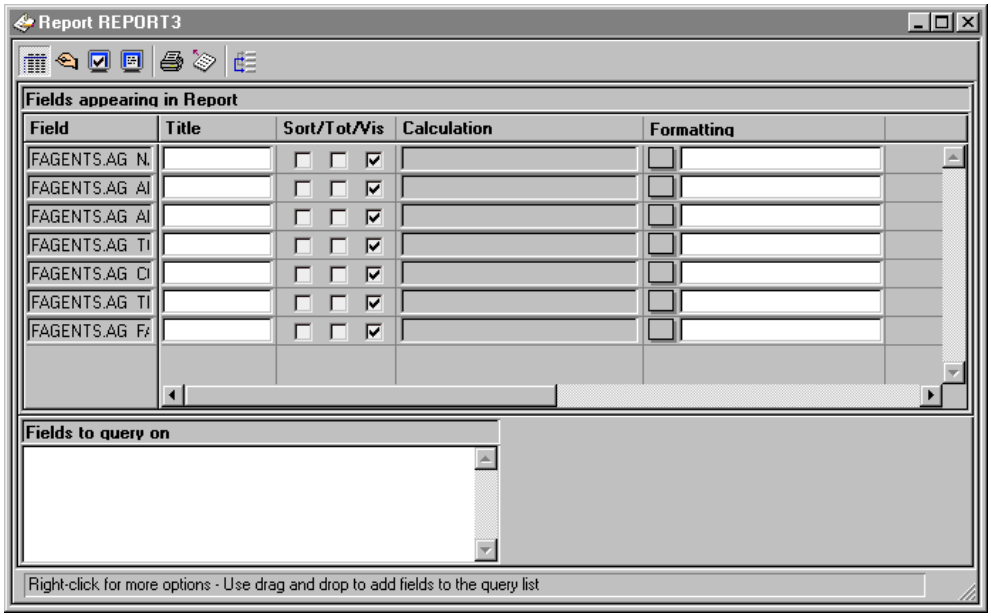


For all templates, you next select the columns you want to include on your report.

- Select the data type of your report by clicking on either the OMNIS or SQL data radio button: the buttons do not appear for SQL sessions
- Open the session or data file name in the list and select the columns you want to include on your report

You can either select a table or file name to include all the columns, or you can open individual tables or files and select individual columns. For OMNIS data access you may need to select the Main file as well.

- When you have selected the columns you need, click on Finish



The Query and field list displays all the columns you included on your report. For each column you can specify a Title or label, whether or not the column is sorted, totaled, or visible, a calculation, and a formatting string. You can select a formatting string from the dropdown list next to the entry field.

At this stage you can print or view the output of your report on screen, but since you haven't added any query or filtering yet you will get back all the data.

To view the output of your report

- Click on the Rebuild Output button, or if this is the first query for the current session, click on the View Output button

The Output screen displays all your data using the layout you selected in the wizard. You can select data from the output screen by dragging the mouse and selecting Copy from the Edit menu, or you can use Select All to select all the data in your report and Copy. This copies your data in tab-delimited format into the clipboard. From the Output screen you can also print directly to the current printer or export the data to a file using the appropriate buttons on the toolbar.

To get back to the Query and field list

- Click on the Query list button in the toolbar

If you attempt to close an ad hoc report without saving, you will be prompted to save before closing.

To save an ad hoc report

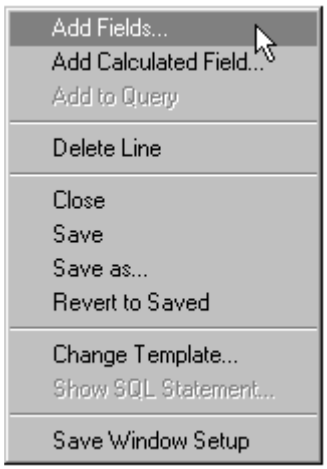
- Right-click on the background of your report and select Save
- Name your ad hoc report, including the .ahr file extension

Adding Columns or Fields

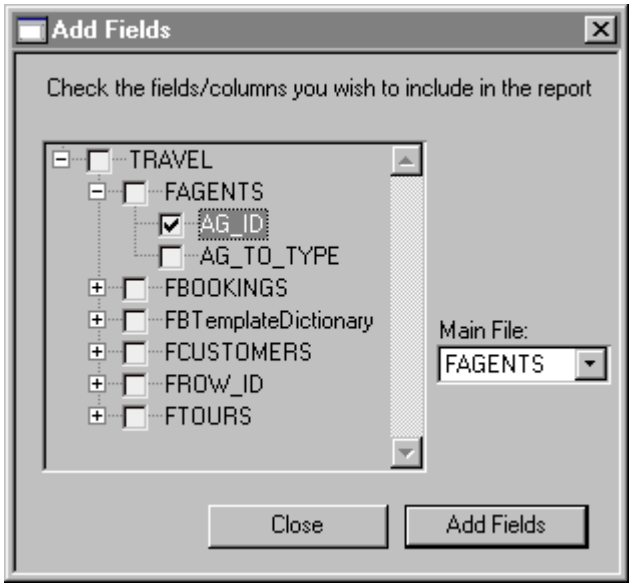
When you create an ad hoc report from scratch you select the columns to include on your report in the wizard, but you can add further columns to your report at any time from the field list context menu.

To add columns or fields

- Right-click on the field list



- Select the Add Fields option from the context menu



- Select the extra columns you want to include on your report, click on the Add Fields button, and click on Close when you're done

For ad hoc reports based on OMNIS data, you can also change the main file from the Add Fields dialog.

Adding and Editing Sort Fields

Assuming you have included all the columns you need in your report, you may want to sort the report in a particular order.

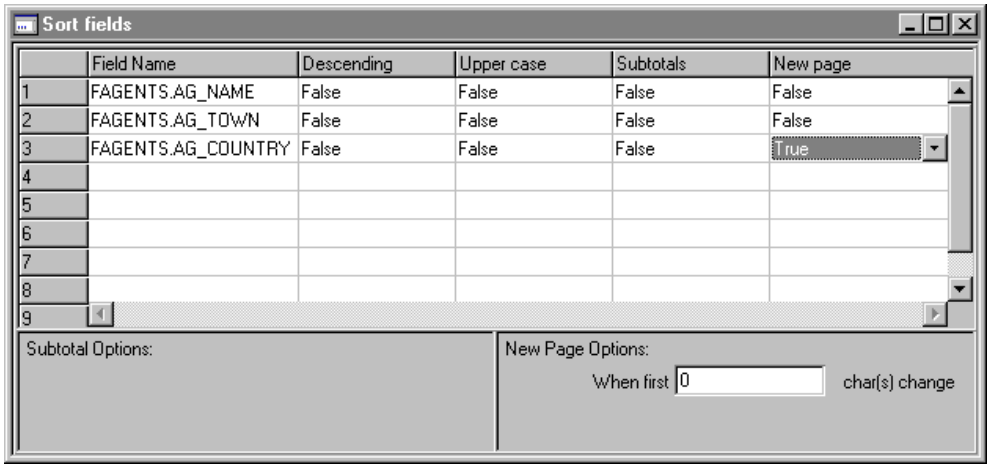
To specify a sort field

- Check the Sort check box for the appropriate columns or fields

The order in which you check the Sort check boxes determines the sort level, that is, the first column you check is the first sort field, the second column you check is the second sort field, and so on. You can specify up to nine sort fields for a single report.

To edit the current sort fields

- Click on the Edit Sort Fields button in the Query and field list toolbar



Note that you must add sort fields in the Query and field list by checking the appropriate box, not from the Sort fields dialog. For each sort field, you can select the following options.

– **Descending**
when set to kFalse the field is sorted in ascending order, otherwise when kTrue the field is sorted in descending order

– **Upper case**
when set to kTrue the data is convert to upper case for the purposes of sorting; not available for SQL data

– **Subtotals**
when set to kTrue triggers the printing of a subtotal section when the sort field changes value; you can specify the number of characters that have to change to trigger the subtotal

– **New page**
when set to kTrue triggers a new page when the sort field changes value; you can specify the number of characters that have to change to trigger a new page

When you rebuild the report, your data is sorted using the specified sort fields in their correct order. For example, you could sort a Customer file on COUNTRY which is sort field 1, on STATE/COUNTY which is sort field 2, and TOWN which is sort field 3.

Adding a Query

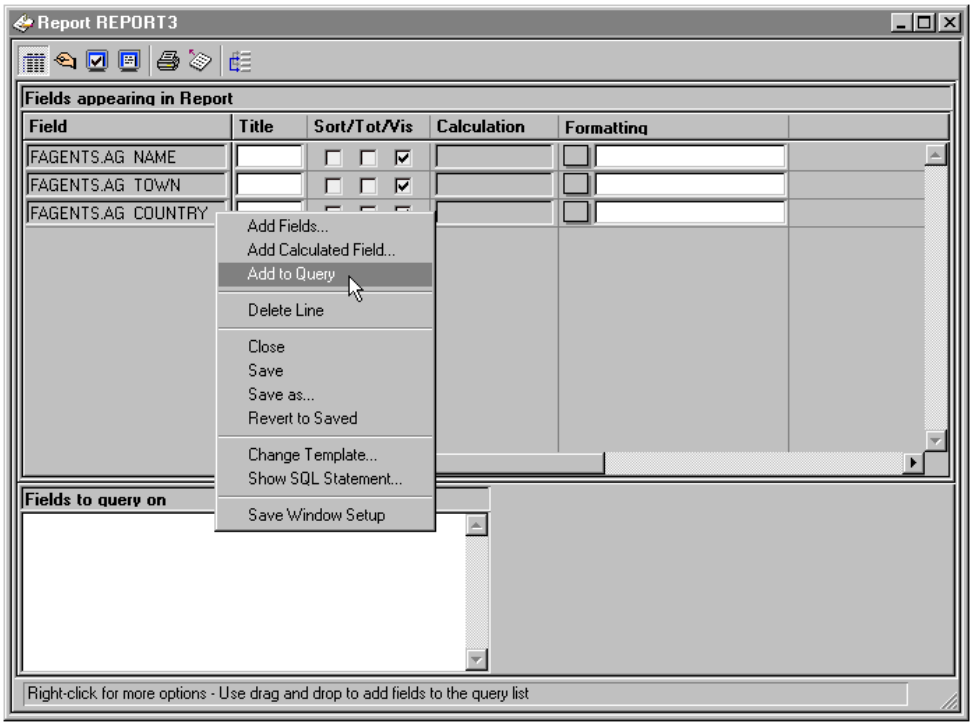
For each column or field in your report you can specify a query that must be fulfilled if a particular row of data is to be included in the output.

To add a query to your report

- In the Query and field view, drag a column or field from the field list into the query list at the bottom of the ad hoc window

or

- Right-click on the column or field in the field list and select Add to Query from the context menu



The column or field is added to the query list. The Equal to comparison is selected by default, but you can select a different operator and enter the value or calculation for the comparison. The following operators are available

- >=
  the data in the column or field must be *greater than or equal to* the specified value
- <=
  the data in the column or field must be *less than or equal to* the specified value
- <>
  the data must be *not equal to* the specified value
- =
  the data must be *equal to* the specified value
- In
  the data must be *In* the specified set of values, for example, to test if the field iColor is set to one of the values red, blue, or green use the query:
  iColor IN 'red','blue','green'; note this is for SQL data only
- Not In
  the inverse of the In clause, that is, the data must *Not* be *In* the specified set of values; note this is for SQL data only
- Contains
  the data must *contain* the specified string value
- Begins with
  the data must *begin with* the specified string value
- Null
  the data in the column or field must be *null,* that is, a value has not been entered
- Not Null
  the data in the column or field must be *not null,* that is, a value has been entered
- Like
  the data must be *Like* the specified string, which can include the following wildcards:
  _ (underscore) to represent a single character, or % to represent multiple characters
- Not Like
  the data must be *Not like* the specified string, which can include the wildcards described above
- Between
  the data must be within a range or *Between* the specified values, for example, to select values between F and G use the query: iField Between 'F','G'; note this is for SQL data only

– Not Between
the inverse of the Between clause, that is, the data must *Not* be *Between* the specified values; note this is for SQL data only

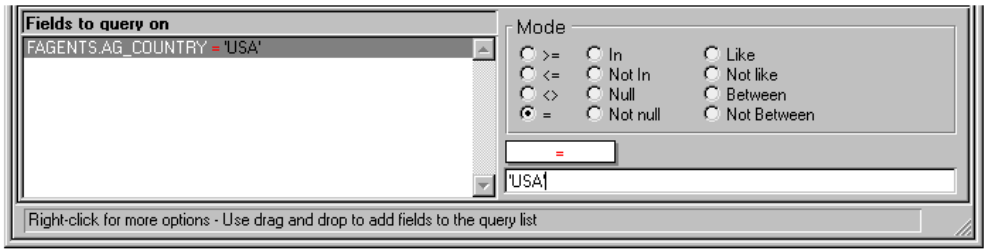
– Child of
the current record must be a *Child of* the specified parent file name; OMNIS data only

– Not child of
the current record must *Not* be a *Child of* the specified parent file name; OMNIS data only

– Parent of
the current record must be a *Parent of* the specified child file; OMNIS data only

– Not Parent of
the current record must *Not* be a *Parent of* the specified child file; OMNIS data only

For example if you want to select all the rows or records from a customer database where the COUNTRY is equal to ‘USA’ you can enter the following query.



Strings must always be quoted, and numeric columns or fields can include numbers or calculated values including other variable names. Boolean fields take values of YES, 1, NO, 0, '' (empty), or Null. You can enter date and times in the comparison field but its interpretation depends on the data format of the column or field.

Multi-line Queries and Logic

You can add multiple columns or fields to a query. A multi-line query always assumes the connecting logic is AND, but you can insert your own logic into the query.

To add logic to your query

- Right-click on the query list at the bottom of the ad hoc window
- Select Insert AND or Insert OR as appropriate from the context menu

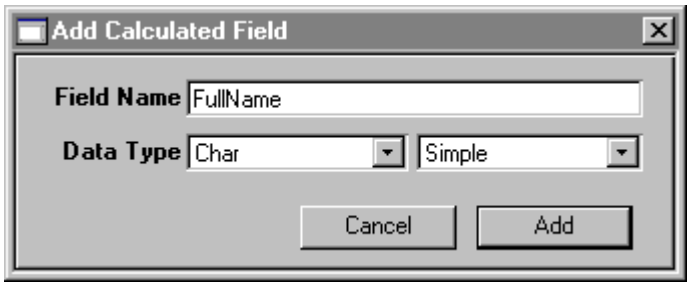
You can drag and drop lines in a multi-line query to reorder the query, including the AND and OR operators. You can also delete a single line or all lines using the query list context menu. You can group particular lines in a multi-line query by selecting the AND and OR lines and clicking on the promote and demote buttons marked < and >.

Adding Calculated Fields

You can add calculated fields to the ad hoc report field list, for example, you can concatenate two values into one report line. The calculations are based on the current values in the columns taken from the current data row or record.

To add a calculated field

- Right-click on the Query list
- Select the Add Calculated Field option from the context menu



- Type the name of the calculated field, select the type and subtype, and click on the Add button
- Enter the calculation in the Calculation box in the field list

For example, in a label report you can enter a calculated field called FullName and use the *con()* function to concatenate the First name and Last name fields.

- Right-click and select Save from the context menu to save your report
- To see the results, click on the Rebuild Output button

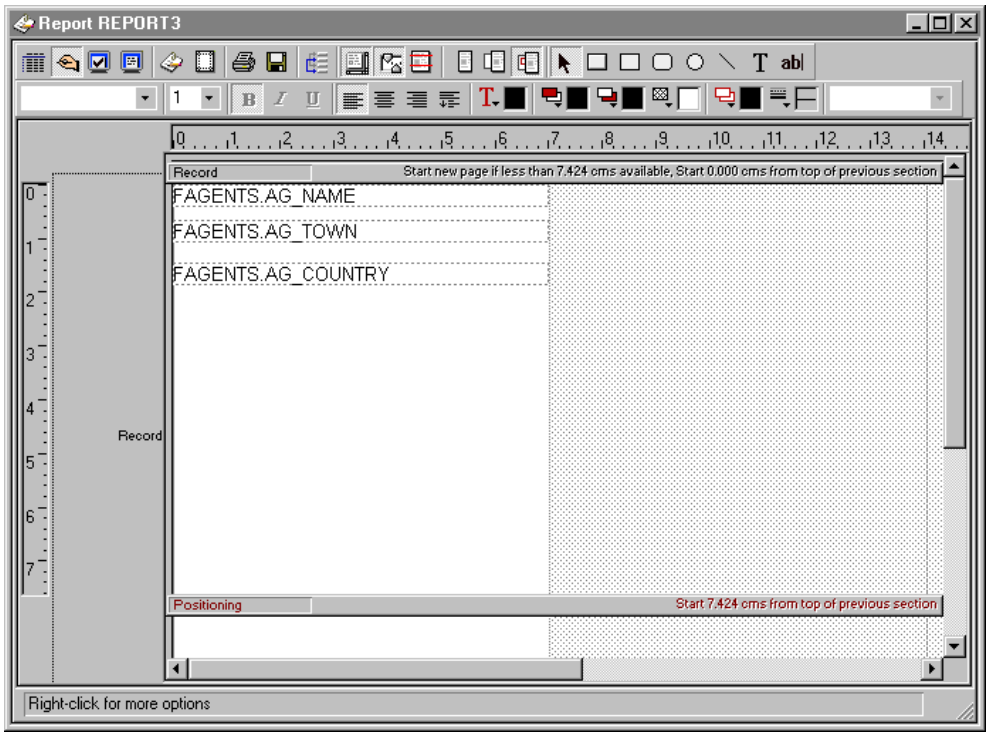
At this stage you may want to change the layout of your report, or add some graphics. Furthermore if you have added calculated fields to your report you may need to rearrange the fields on the report.

Modifying your Report

You can change the layout and positioning of the fields on your report at any time by switching the ad hoc window to modify report mode.

To modify your report

- Click on the Modify Report button in the ad hoc window tool bar



You can add text labels and graphics, and move fields around and change their colors. If you delete columns or fields from the report they will not appear in the output, but they will still be included in the field list and are still part of your query.

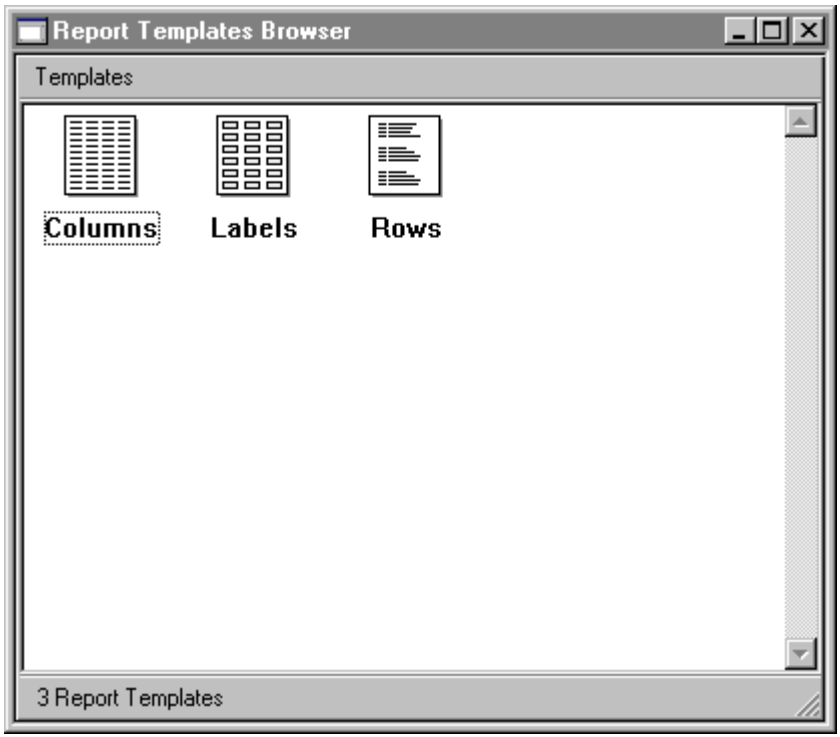
- When you have made changes to the report modify window you can Right-click and select Save to save your report

Ad hoc Report Templates

When you create a new ad hoc report you base it on one of the templates stored in the ad hoc library. However you can change the templates via the Template Browser.

To change a report template

- Select Template Browser from the Reports menu



- Double-click on the template you want to change

You can add graphics and change the font and color properties of any object in a template, and you can examine the methods behind the report. However you should avoid changing the structure of the sections and programming in the templates provided. The Templates menu in the Template Browser lets you create your own templates based on standard report classes. Your own templates will appear in the Ad hoc Report Wizard when you next create a new report.

Re-using Ad hoc Reports

Ad hoc reports are saved to disk as small library files with the .ahr file extension. Each file stores a copy of the report layout and query. You can open an existing ad hoc report file from the Reports>>Open menu option.

Ad hoc report Notation

The public methods $x_newadhoc and $x_openadhoc are available in the ad hoc report tool. You can call a public or tool method using the $dotoolmethod() method, available under the $root.$modes group. $x_newadhoc prompts the user to create a new ad hoc report by launching the ad hoc report wizard, and $x_openadhoc prompts the user to open an existing ad hoc report. These methods do not take any parameters as follows

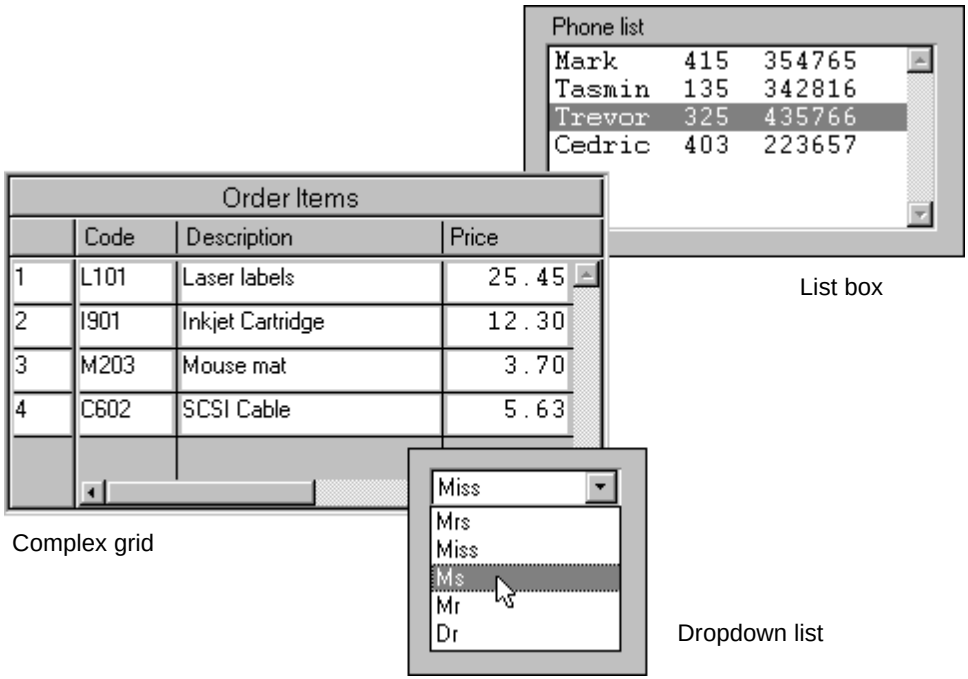
```
Do $root.$modes.$dotoolmethod(kEnvToolAdhoc,'$x_newadhoc')
Do $root.$modes.$dotoolmethod(kEnvToolAdhoc,'$x_openadhoc')
```

The table selection window by default has a checkbox option to display tables for all users if the database supports this feature. You can hide and show this option using the $x_allowanyuseroption() method as follows

```
Do $root.$modes.$dotoolmethod(kEnvToolAdhoc,
   "$x_allowanyuseroption",[kTrue|kFalse])
```

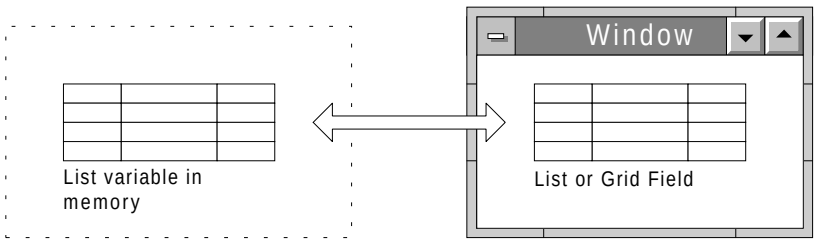
Chapter 10—Lists and Grids

Lists and Grids are types of window field in which you can show all types of data including multiple columns and rows. OMNIS provides many different type of grid and list field for you to display list data. For example, in an accounts application you could use a complex grid field on the order processing window to enter or display the individual items for a purchase order. In a customer database, you could use a dropdown list to show the salutation as a short list of preset choices from which the user selects a value. And you can use a simple list box to show names and phone numbers.



In the Component Store you can choose from many different types of list or grid field, including, popup lists, dropdown lists, combo boxes, tree lists and string and data grids. Short single-column lists are suitable for selecting a value from a small number of choices, while multi-column, multi-row grids are more suited to complex data entry and manipulation.

The data which is displayed in a list or grid field is taken from a *list variable*. To create a list or grid field, you first declare a variable of list data type and then populate it. Usually this involves writing a method to transfer the data from a database into the list variable. When you place the list or grid field on a window you make its dataname the name of your list variable. In this way there is a direct link between the data in the window field and the list in memory.



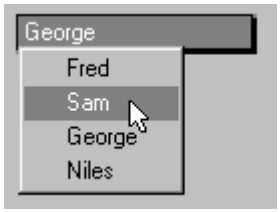
You can write methods behind the field to respond to user clicks in the list or grid field on the open window, and you can drag and drop data to and from a list field.

This chapter describes the various types of list and grid fields available in OMNIS. It also describes how you create and define list variables, and how you change the visual appearance of grid fields using the Property Manager. The manipulation of lists and complex grids is described in the *List Programming* chapter in the *OMNIS Programming* manual.

Types of List and Grid Field

The Component Store contains many different types of list and grid field. Each one is suited to a particular type of data.

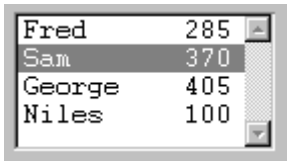
- **Popup list**
  displays the currently selected value: best suits a short list of commonly used values



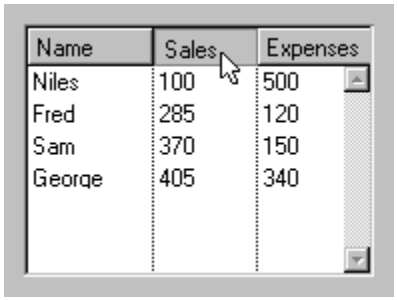
- **Dropdown list**
  lets you select a value from the dropdown list: best suits a short list of commonly used words or values

- **Combo box**
  lets you enter a value into the field or select a value from the dropdown list

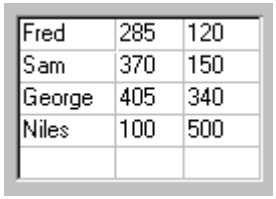
- **List box**
  can display a number of columns of data, but has no column dividers



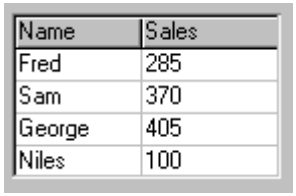
- **Headed List Box**
  presents the data in tabular format with button style column headers; you can program the list to sort when you click on a header; the same type of control as used in the Browser Details view



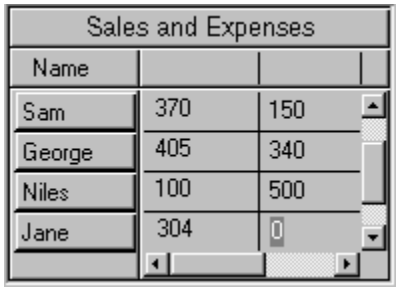
- **String grid**
  presents text data in tabular format and lets you enter text data into the cells. You can size the columns by clicking on the header buttons



- **Data grid**
  similar to a string grid, but supports most data types

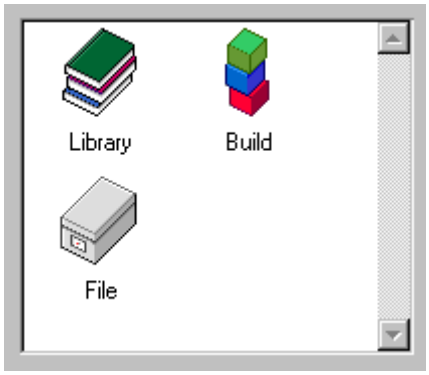


- **Complex grid**
  presents data as a series of columns and rows similar to a spreadsheet; in runtime the user can create new rows, enter data into cells, and size the columns

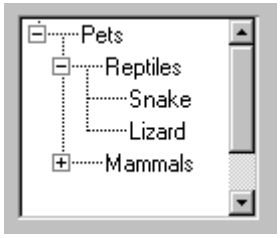


- **Check list**
  displays a single column of data and check boxes which you can check and uncheck

- **Icon Array**
  presents a number of icons with text labels either in large or small icon view; the same type of control as used in the Browser and Component Store



- **Tree list**
  presents a hierarchical list of values; you can expand and collapse each node to show or hide the next level; the same type of control as used in the Notation Inspector



Before you can create a list or grid field on a window, you must declare and define a list variable. Then using a method, you make your list variable the current list, define the columns in the list, and build the list by inserting data, usually by selecting it from your server or OMNIS database. Finally you create a list or grid field on your window and assign the list variable to the field.

The following sections describe how you create a list variable and define the columns for your list. The remainder of the chapter describes how you create the simpler types of list box and grid field: the complex list and grid types are described in the *Window Programming* chapter in the OMNIS Programming manual.

List Variables

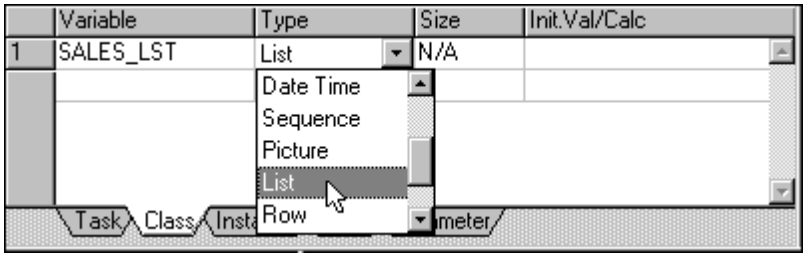
A *list variable* is a structured data type that can hold multiple columns and rows of data. You can declare a task, class, instance, local, or parameter variable of list data type. You can create any number of list variables in your application, limited only by the amount of memory in your system. A list can hold an unlimited number of lines and can have up to 400 columns. The data type of each column in your list can be any one of the OMNIS data types including Character, Number, Date, Picture, and List:

Typically, when creating and building a list using OMNIS commands, you

1. Declare a variable in the method editor with the list data type
2. Make this list variable the *current list* with the *Set current list* command
3. Define the field names or variables to use as the columns in the list with the *Define list* command
4. Build your list, normally from your server or OMNIS database

Creating a List Variable

To create a list variable, open the method editor for the class in which you want to create the list variable. For example, if you want to create a list or grid field on a window open the class methods for window class. In the method editor variable pane, create a new variable and give it a name. Select the List data type.



Typically you then write a method that sets the current list, defines the columns for your list, and builds the list. This method is best placed in the $construct() method of your window or placed elsewhere and called from the $construct() method. This ensures the list is built just before the window opens.

Setting the Current List

A number of the commands make an assumption that the list you are referencing is the current list. To make your list the current list you use the *Set current list* command. For example, using the list variable SALES_LST your method would start

```
Set current list SALES_LST
```

Defining Your List

Next you specify the column structure for your list using the *Define list* command. This command takes a list of variable or data names as parameters.

```
Define list { NAME, SALES, EXPENSES }
```

This command means that the variables NAME, SALES, EXPENSES will be included in your list. Their names and data type will define the names and data type of each column in your list. The variables used in your list can be from anywhere in your library, including being defined as variables in the class containing your list.

You can include all the columns from a file class in your list by using just the name of the file class in the *Define list* command. You can also define a list from a schema, query, or table class to directly map the columns of your list to the columns in your server database. In this case you can use the *Define list from SQL class* command which is discussed in the *List Programming* chapter in the *OMNIS Programming* manual.

Building Your list

To populate your list variable with data you need to build the list using one of the following commands.

– *Build list from select table*
  builds a list from a server table

– *Build list from file*
  builds a list from the main file class in an OMNIS data file using the specified indexed field

– *Add line to list { (value|field1, value|field2,...) }*
  adds a new line to the current list using the specified values, or the current values of the specified fields

The complete method for defining and building a list from an OMNIS data file could be

```
Set current list SALES_LST
Define list { NAME, SALES, EXPENSES }
Build list from file on NAME
```

Or if you want to build a static list using *Add line to list* your method could be

```
Set current list SALES_LST
Define list { NAME, SALES, EXPENSES }
Add line to list {('Fred',285,120)}
Add line to list {('Sam',370,150)}
Add line to list {('George',405,340)}
Add line to list {('Niles',100,500)}
```

Creating List and Grid Fields

You can create many different types of list and grid fields. Some types are most suited to single columns of data while others can contain multiple columns and rows and are best for complex data entry. In the introduction to this chapter you have seen the different types of grid available in OMNIS. The following sections describe how you create some types of list and grid field, including a single-column *dropdown list* with default lines, a *list box* with two columns, and a *complex grid* with many columns and rows.

Dropdown lists

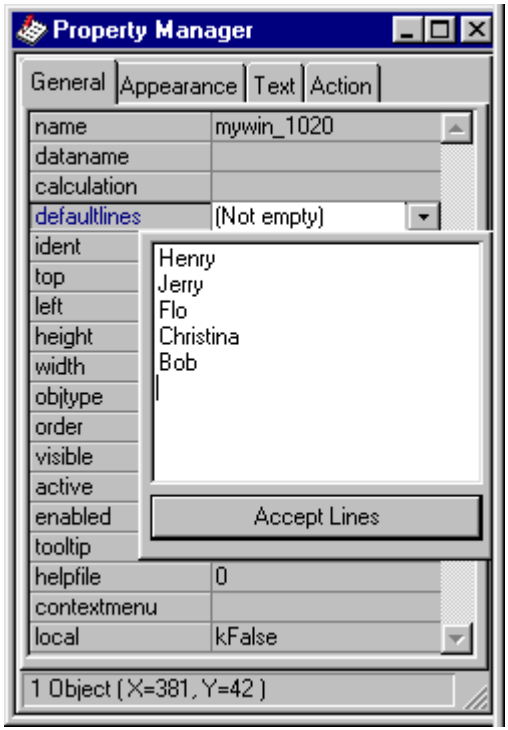
The simplest type of list or grid field contains a single column of data. All types of list or grid field can contain a single column, but some types such as list boxes, dropdown lists, combo boxes, or check box lists are best suited for displaying short single columns of data. The data inside a short list or grid field can come from your database loaded into a list variable, or you can include a number of lines that always appear in the list by default.

To create a dropdown list with default lines

- Drag a *Dropdown List* from the Component Store onto your window
- Bring the Property Manager to the front using F6/Cmnd-6
- Click on the **defaultlines** property and click on the entry field

A small entry box will appear.

- Type a list of values you want to appear in your dropdown list



- Click on Accept Lines
- Open your window and click on the list

When you click on the dropdown list the default lines will appear. You can add default lines for dropdown, combo box, and tree list fields.

List boxes

If you want to include two or more columns in your list you have to get your data from a list variable, since the **defaultlines** property can handle single columns only. When you create a list field you have to specify the list variable associated with the field.

To create a list box

- Drag a *List box* from the Component Store onto your window
- Bring the Property Manager to the front using F6/Cmnd-6
- Enter the name of your list variable in the **dataname** property for the list box field
- Enter the **calculation** in the format

```
jst(column1,width,column2,width,...)
```

The **calculation** for a list box field specifies the column(s) to appear in the grid and the width of each column as the number characters, using the *jst()* function. You can include in your list field any number of columns from your list variable, and in any order.

The jst() function

The calculation for your list field can use the *jst()* function to set the column names and their widths. For example, you could enter

```
jst(NAME,10,SALES,5)
```

This calculation puts the NAME variable in a column 10 characters wide and the SALES variable in a column 5 characters wide.

You can include the X modifier to truncate the data in a particular column. For example

```
jst(COMPANY, '20X', NAME, '22X')
```

This calculation puts the COMPANY variable in the first column and truncates any data that is longer than 20 characters. It then puts the NAME variable in the second column and truncates any data that is longer than 22 characters.

You can right justify a column, perhaps one containing numbers. For example

```
jst(NAME, ‘20X’, TEL, -18)
```

This expression left justifies the NAME variable in the first column which is 20 characters wide, and right justifies the value of the TEL variable in a column 18 characters wide.

Note that for single-column lists you need only include a single field or variable name in the **calculation** property, that is, you don’t need to use the *jst()* function. If you omit the calculation altogether, no data will appear in your field.

The default properties of a list box are set, but you can change any of these in the Property Manager.

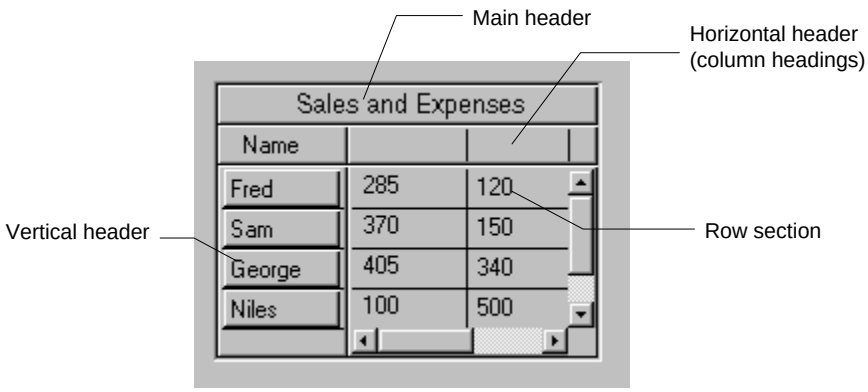
You may need to adjust the column widths or the text properties. For example, you should use a non-proportional font such as Courier if you are using two or more columns in a list box. This ensures the columns line up across the list.

When you select a line in a list box, this normally deselects all other lines, but with the **multipleselect** property set, you can select any number of lines. Dragging or shift-clicking selects contiguous lines, while Ctrl/Cmnd-clicking selects non-contiguous lines. You can deselect individual lines with Ctrl/Cmnd-click or all lines by clicking in the white space at the end of the list.

If you open your window and the list or grid field is empty, this probably means that either the list variable behind the field is empty, or you have not entered the **dataname** or **calculation** property correctly.

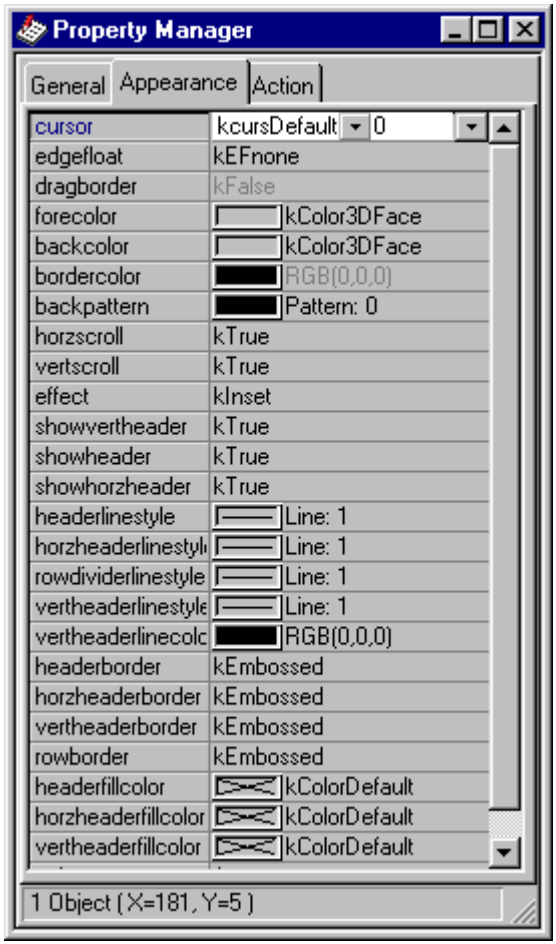
Complex Grids

A *complex grid* is a field that contains other fields and displays data from a list variable. You define the columns for a complex grid by dragging fields from the component store into the *row section* of the grid field. The fields in a complex grid should correspond to some or all of the columns in the list variable supplying the grid data. Complex grids also have three header sections which are visible by default: the *main header*, the *horizontal header* for column headings, and the *vertical header* for row headings. You can add text and graphics to any of the header sections. Any background object you put in the row section of a grid, such as a line or graphics, will be repeated for each row of the grid.



To create a complex grid field

- Drag a *Complex Grid* from the Component Store onto the window
- Bring the Property Manager to the front using F6/Cmnd-6
- Enter the name of your list variable as the **dataname**
- Click on the **Appearance** tab



The majority of the grid properties hide or show parts of the grid, including the various headers. For example, you can control the style, color, and line style of the main header using the **showheader, headerlinestyle, headerborder,** and **headerfillcolor** properties. You can set the style of the row border, and the number of columns to appear in the grid field.

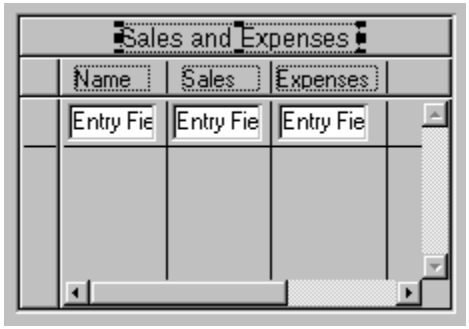
- Click on the **columns** property and set it to the number of columns you want in your grid field
- For each column in your grid, drag an Entry field from the Component Store into the appropriate column in the row section of your grid field; release the field when the cell border highlights

- With the column field selected, set its **dataname** to the name of a column in your list variable behind the grid field
- Do this for each column in your grid field

You can also set the font, style, alignment and color for each column field. In a similar way, you can add a text label to each column header. To do this

- Drag a **Text** object from the Component Store into each column heading and the main header section; release the field when the header border highlights
- With the object selected, type the header text in the **text** property
- For each text object you can set the font, style, alignment, and color

For a three-column grid, your field might look something like this.



You can make a complex grid **enterable**, so the user can enter data into each cell, and **extendable**, so that tabbing out of the last cell creates a new row. For complex grids and some other types of list and grid field you can make the field display multiple selected lines by enabling the **multipleselect** property. These properties are on the Action tab in the Property Manager. In general, the tabbing order of the fields inside a table starts from the header, then to the horizontal and vertical headers, and finally to the rows. This overrides the field order of the normal field numbers.

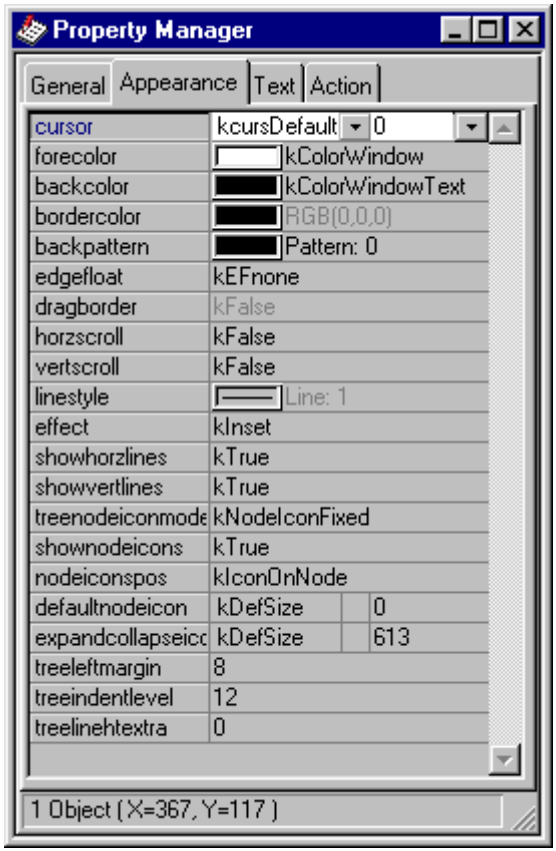
For details on programming a complex grid field refer to the *Windows Programming* chapter in the *OMNIS Programming* manual.

Tree Lists

The tree list displays a hierarchical list which you can expand and collapse to show or hide the different levels. The Notation Inspector uses such a list to display the notation tree. A tree list can either display a list variable or you can enter a default list.

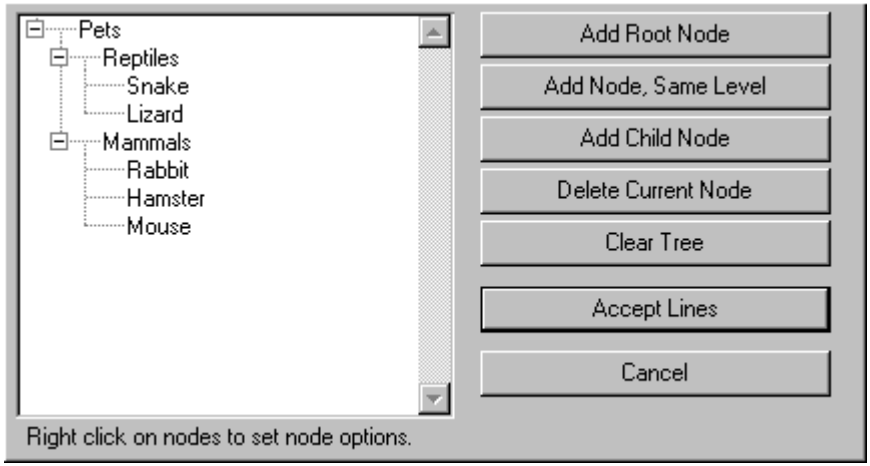
To create a tree list with default lines

- Drag a *Tree List* from the Component Store onto the window
- Bring the Property Manager to the front using F6/Cmnd-6
- Click on the **Appearance** tab



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

- Click on the **treedefaultlines** property on the General Tab and click on the entry field



The default lines box lets you enter a tree by adding root and child nodes. The tree list contains a number of options, including populating a tree list from a list variable, and these are described in the *List Programming* chapter in the OMNIS Programming manual.

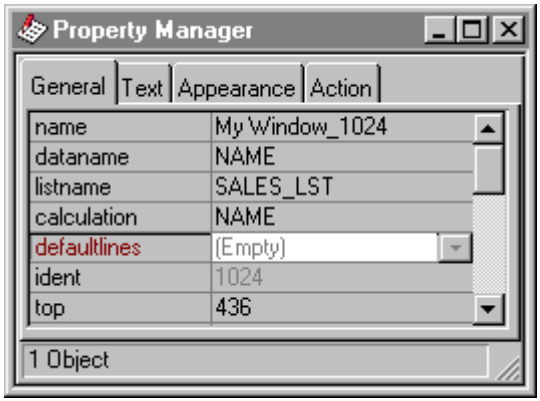
Combo Boxes

A combo box is a combination of dropdown list and an entry field. When viewed on an open window, you can choose an item from the list or type anything you want into the entry box. You might therefore put the salutations *Mr, Mrs, Ms, Miss* in the list, and leave the user free to enter *Dr* when needed. You can open a combo by clicking on the drop arrow, or tabbing to the field and pressing Alt plus the down arrow key; pressing just the down arrow key cycles through the choices in the list. Selecting a line with the mouse or pressing the Tab or Esc key closes the list. When you create a combo box you must specify the name of the data field in the **dataname** property and the name of your list variable in the **listname** property.

To create a single-column combo box

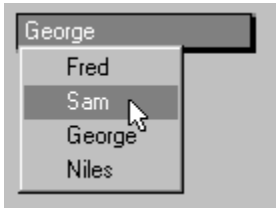
- Drag a *Combo Box* from the Component Store onto the window
- Bring the Property Manager to the front using F6/Cmnd-6
- Enter the **dataname** for the entry field part of the combo box, that is, the column name of the field or variable in your list
- Enter the name of your list variable in **listname**

- Enter the **calculation** for the list; this can be any calculation to format the list for display



When using data from a list you should leave the **defaultlines** property empty. Some of the properties of a combo box refer to the entry field part. On the Appearance tab, you can specify the number of lines displayed when your combo box drops down by setting the **listheight** property.

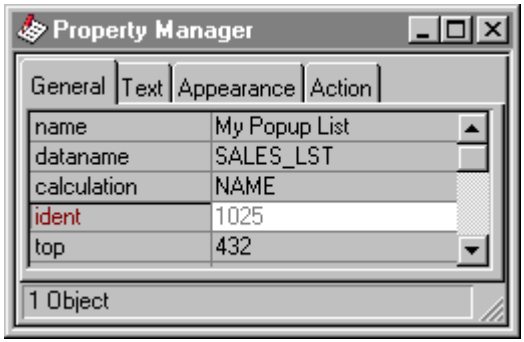
Popup Lists



Popup lists are most suited to short single-column lists from which the user can select a single choice. The user can click on the field to drop down the list. When you create a popup list you need to enter the name of your list variable in the **dataname** property for the field. Enter the name of the variable in your list column in the **calculation** property. Use the constant kDefaultBorder for the **effect** property to ensure the list has the default border style for the current operating system.

To create a popup list

- Drag a *Popup List* from the Component Store onto your window
- Bring the Property Manager to the front using F6/Cmnd-6
- Enter the name of your list variable in **dataname**
- Enter the **calculation** for the list, or a variable name for a single column list



Check Lists

Check lists display a check box against each row in the list field. The user can select a line by checking the check box for a line. They are most suited to single columns of data or values in which the user can check multiple values or choices. If a check box is checked the corresponding line in the underlying list variable is selected.

To create a check list

- Drag a *Check List* from the Component Store onto your window
- Bring the Property Manager to the front using F6/Cmnd-6
- Enter the name of your list variable in **dataname**
- Enter the **calculation** for the list, or a variable name for a single column list

Getting Data from a List or Grid Field

You can use lists and grids to display information, but typically you want the user to choose a particular line in a field and find out which line they have selected. Using a method behind the field you can load the data in the selected line, find out the row number of the selected line, or for complex grids find out the data in a particular cell.

To create a list field method

- Right-click on the grid field in the design window
- Choose Field Methods from the context menu

or

- Double click on the grid field
- Select the $event() method

When you open the method editor you get the event handler for the field

```
On evClick        ;; Event Parameters - pRow ( Itemreference )
```

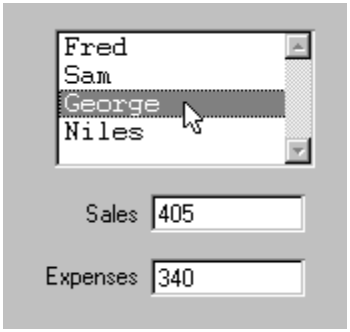
You can enter any commands you want in this method to respond to clicks and you can add code to deal with other events, such as double-clicks. If the list columns are defined from file class or other declared variables, you can transfer the current values of the fields or variables in the selected line to the CRB using the *Load from list* command. The method behind your list field might be

```
On evClick
   Load from list
   ; now do something with the values
```

See the *Methods and Notation* and *Events and Messages* chapters in the *OMNIS Programming* manual for more details.

Lists and Local Fields

You can place a series of *local fields* on your window that correspond to the columns in your list variable, so that when the user clicks on the list or grid the current list values are loaded into these fields. In this case the **dataname** of each local field should be set to the name of a column in the list variable, and their **local** property should be set. The field numbers of the local fields must immediately follow the grid field. To set the field order of a field change its **order** property.



When you click on a line in the grid the local fields will update automatically.

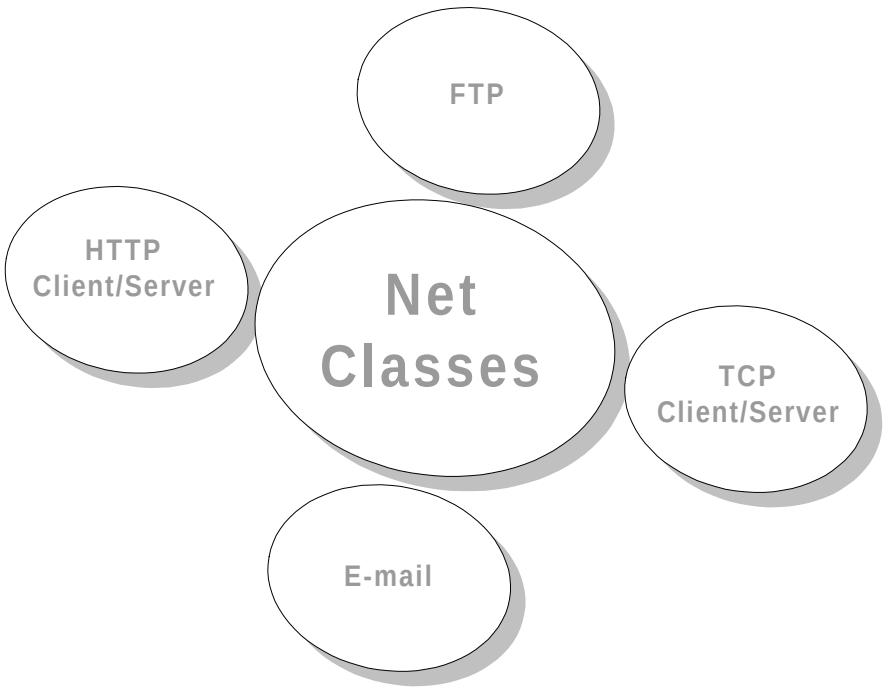
Searching in List and Grid Fields

All OMNIS list and grid fields have a built-in search facility. At runtime, you can click on a list or grid field and type the first few characters of the text you want to find, and OMNIS will try to find a line containing the text. For example, you can tab to a list field and type “b” to find the first line *beginning* with “b”. You can use an asterisk “*” to represent any number of arbitrary characters, so “*s” searches for the first line that *contains* the character “s”, and the string “s*e*n” finds the line starting with “s” and containing “e” and “n”. OMNIS beeps if it cannot find a matching line. You can use the + and - keys to find the next or previous line in the grid that matches the string.

For a complex grid you can specify which column the search applies to. To do this, enter the name of the column in the **calculation** box when you create the grid field. For this to work in a grid field you must turn off its **enterable** property. Otherwise when you type on a line your search the value will be entered into the grid.

Chapter 11—Internet Classes

OMNIS provides a number of powerful Internet-ready object and code classes, and various easy-to-use window wizards that let you fully integrate Internet services into your OMNIS applications. The Net classes and wizards let you create windows to send and receive E-mail, download files from a remote server using FTP, and access information and services on an HTTP server via the Web.

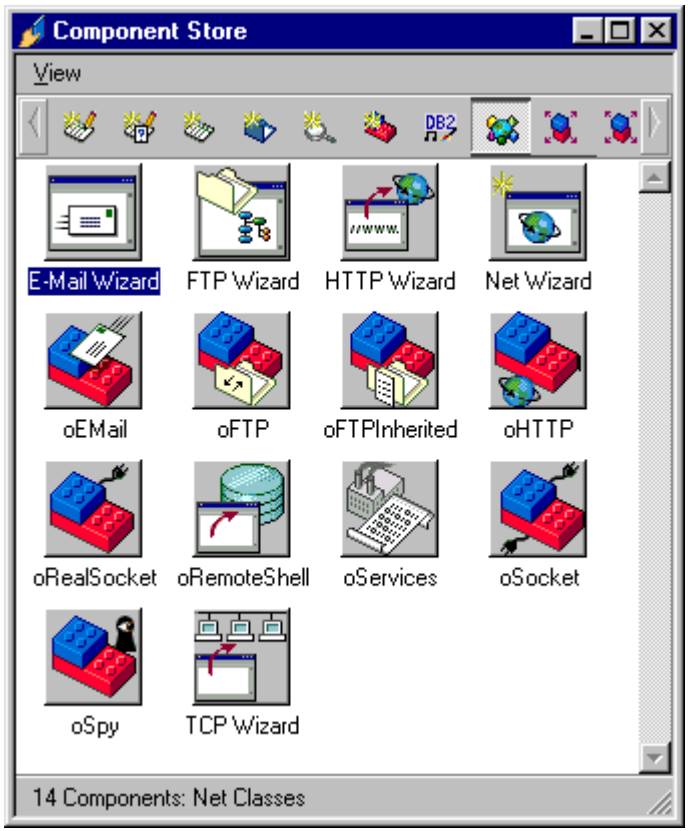


The OMNIS object and code classes, and the window wizards are available in the Component Store under the Net Classes button. You can copy the object and code classes to your own applications and use their methods, or you can use the window wizards to create ready-made windows for your application. When you drag one of the window wizards from the Component Store onto your library the object and code classes it needs are copied across to your library automatically.

For general information about integrating Internet services into your OMNIS applications, and using object classes, refer to the *Internet Programming* and *Object Oriented Programming* chapters in the *OMNIS Programming* manual.

To view the Net window wizards

- Open your library and press F3/Cmnd-3 to open the Component Store
- Scroll the toolbar at the top of the Component Store and click on the Net Classes button



The Net Classes group in the Component Store contains the Internet-enabled window wizards and a range of OMNIS object classes that form the OMNIS Net objects API. The Net window wizards include

Net Wizard
lets you create any of the Net windows including an FTP window, HTTP client or server window, TCP client or server window, or an E-Mail window

FTP Wizard
lets you create a window for accessing files on a remote FTP server

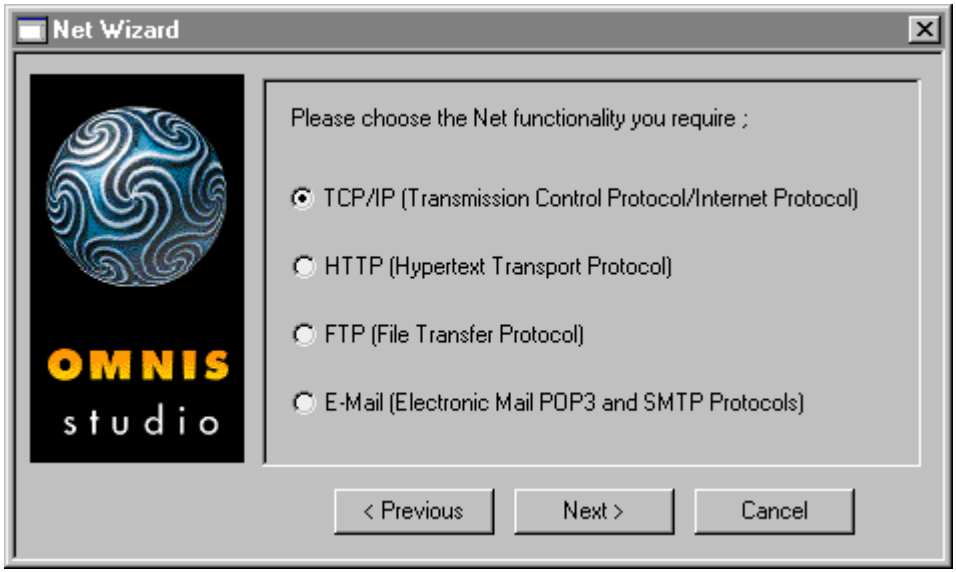
HTTP Wizard
lets you create an HTTP client or server window

TCP Wizard
lets you create a TCP client or server window

E-Mail Wizard
lets you create a window to send and receive e-mail in your application

To create a Net-enabled window

- Drag the Net Wizard component onto your library
- Name the window class, press Return, and at the introductory window press Next



- Choose the functionality you require and click on Next

For the TCP and HTTP windows

- Choose either Client or Server network functionality

For the FTP and E-mail windows you can enter your own details which become the defaults for that particular window. You can create another copy of the same type of window with different defaults, for example, you can create an e-mail window for each person in your department.

For the FTP window you can

- Enter your Username, Password, and Site address, or leave the fields blank

For the E-mail window you can

- Enter your E-mail address, Password, SMTP server name, and POP3 server name, or leave the fields blank

At any stage you can click on the Previous button to go back and change your details, or click on Cancel to halt the wizard process and delete the window from your library.

- When you have entered all your details click on the Create button

OMNIS creates your window and copies across any OMNIS object classes required for the window. The new window is opened in design mode ready for you to modify. Alternatively you can press Ctrl/Cmnd-T to open or test your window now. The name and address details you entered in the wizard will be copied to your window automatically.

Net Objects API

You can use the OMNIS object and code classes in your application to create your own windows that provide various Internet services. The Net object and code classes, located under the Net Classes group in the Component Store, form the OMNIS Net objects API. They include

oEMail
object class that provides methods for e-mail services


oFTP and **oFTPInherited**
object classes that provide methods for FTP services

oHTTP
object class that provides methods for client and server HTTP services

oRealSocket and **oSocket**
object classes that provide methods for manipulating TCP/IP sockets

oSpy
object class that maintains a list of sockets

oServices
code class that contains general purpose methods for manipulating sockets

You can call a method in an object class using the notation VarName.$MethodName(), where VarName is an instance variable based on the object class and $MethodName() is any method defined in the object class. The windows created using the window wizards contain instance variables based on the Net object classes.

To view the variables and methods for a Net object class

- Copy the Net object class to your library from the Component Store
- Double-click on the object class in the Browser

or

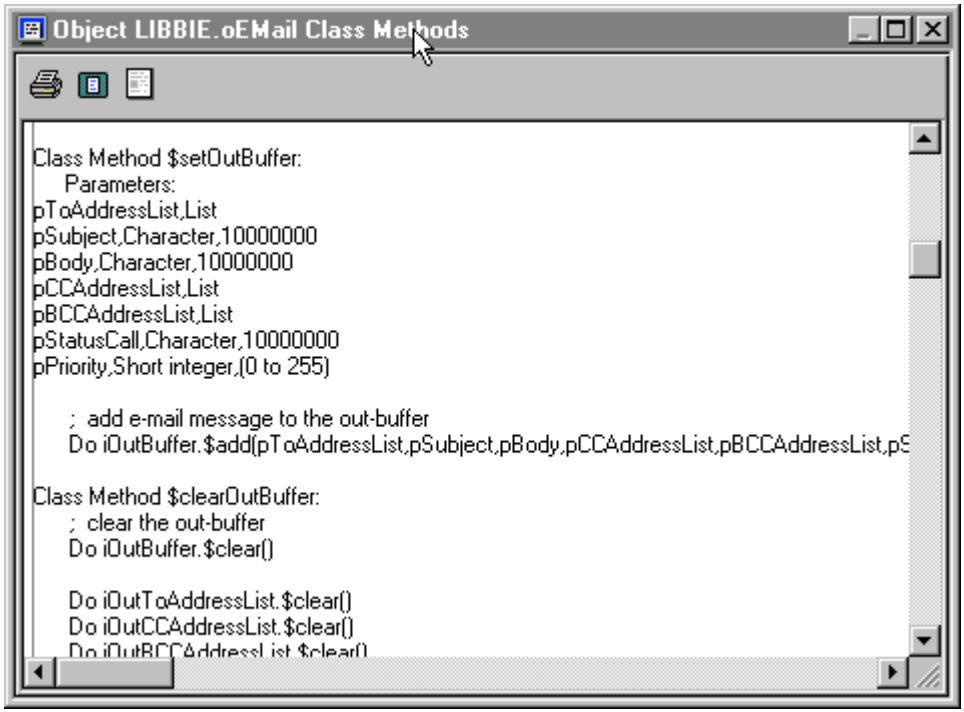
- Right-click on the object class and select the Methods option from the context menu

You can print a list of the variables and methods in any class, including the Net object classes, using the Print Method option in the File menu. You can print the selected method only or, if you click in the white space in the method list to deselect the current method, you can print all the methods for the current class. The report is sent to the current report destination (File>>Print Destination option) and lists the parameters, variables, and code for each method, or for all methods in the class.

To print the methods for a Net object class

- Set the print destination using the File>>Print Destination menu option on the main menu bar
- Select File>>Print All Methods (or Print Selected Method) from the main menu bar

For example, the following screenshot shows the $setOutBuffer() method contained in the oEMail object class. The print out shows you the method's code and its parameters, including variable types and lengths.



The following section describes the Net object classes and summarizes their methods. Some of the Net classes contain empty methods or placeholders for further development; these methods are not listed below.

E-Mail Object Class

The oEMail object class uses the SMTP and POP3 protocols and lets you send and receive e-mail. You can log onto your SMTP mail server, handle the In and Out buffers, and query the mail server for incoming mail. The object class has the following methods.

Method	Description
$logon(pOutServer,pOutFrom,pOutFromName, pInServer,pInUsername,pInPassword)	logs on to your mail server
$setOutBuffer(pToAddressList,pSubject,pBody, pCCAddressList,pBCCAddressList, pStatusCall,pPriority,)	adds e-mail message to the out-buffer
$clearOutBuffer()	clears the out-buffer
$sendMessage(pToAddresslist,pSubject,pBody, pCCAddresslist,pBCCAddresslist, pStatusCall,pPriority)	sends internet e-mail message via a SMTP server
$sendAllMessages()	sends all e-mail messages held in the out-buffer via a SMTP server
$setInBuffer(pReturnPath,pDate,pPriority, pFrom,pSubject,pBody)	adds e-mail message to the in-buffer
$clearInBuffer()	clears the in-buffer
$receive(pDelete,pStatus)	receives e-mail messages
$stat()	finds out how many e-mail messages are waiting to be retrieved
$split(pMessage,pReturnPath,pDate,pPriority, pFrom,pSubject,pBody)	splits the header information from the body of the e-mail message
$logoff()	logs off and clears the buffers

FTP Object Classes

The FTP object classes give you all the methods required to query and access files on a remote FTP server. You can navigate the remote server and display its full directory structure. You can download Ascii text and Binary files and query their read/write and ownership status. The oFTP object class has the following methods.

Method	Description
$construct()	sets up the Socket
$connect(pServerAddress,pUsername,pPassword)	establishes an ftp connection
$disconnect()	disconnects the server
$getWorkingDirectory()	finds the present working directory
$setWorkingDirectory(pNewDir)	clears the current directory and establishes the specified directory
$rename(pOldName,pNewName)	renames the file or directory
$getFileProtection(pDirList,pDirPath)	returns the full directory list including protection rights
$setFileProtection(pFileName,pMode)	sets the protection rights for the specified file
$getLastStatus()	finds and returns the last ftp error
$setFileMode(pFileMode)	sets the file mode to ascii text (kFalse), or binary (kTrue)
$getDirectoryList(pDirList,pDirPath,pMode)	returns directory list
$getFile(pRemoteFile,pLocalFile,pLocalBinary)	returns the specified file

The oFTPInherited object class inherits all the methods from the oFTP object class and has the following methods of its own.

Method	Description
$reconcileDirectoryList(pFullList)	tidies the directory list
$parseDirectoryList(pDirList,pParsedDirList)	parses the directory list
$parsePrivileges(pFilePrivs,pParsedFilePrivs)	finds what priveleges have been set, returns booleans
$stripSeparator(pListIn)	strips excess separators

HTTP Object Class

The oHTTP object class lets you access the World Wide Web, examine web pages, and process files on an HTTP server. It has the following methods.

Method	Description
$get(pHostname,pURI,pCGIList,pPort)	connects to the specified server
$header(pHeaderList)	sends the specified HTTP header
$page(pURL)	reads the specified HTML page
$parse(pMessage,pHeaderList,pMethod, pHTTPVersion,pURL,pCGIList)	parses header info into an OMNIS list
$post(pHostname,pURI,pCGIList, pHeaderList,pPort)	submits a cgi request to a web server
$server(pPort,pConnectSocket)	initializes a server
$splitHTML(pMessage)	parses the html from a web page into an OMNIS list
$splitURL(pURL,pHostname,pURI)	splits URL into Hostname and URI

TCP/IP Object Classes

The Socket object classes let you manipulate TCP/IP sockets. You can open sockets, attach a socket to a specified port, and send and receive messages via a socket. The oRealSocket object class inherits all the methods from the oSocket object class and has the following methods of its own.

Method	Description
$socket(pSocketNum,%0)	sets the current socket or creates a new one
$bind(pPort)	binds the current socket to the specified local port
$listen()	puts the current socket into Listen mode awaiting requests
$accept(pConnectSocket)	accepts the first connection for the socket
$connect(pServer,pService)	opens a new socket to a service or port on the specified server or IP address
$send(pMessage)	sends a message on a socket
$receive()	receives a message on a socket
$close()	closes and releases a socket
$getRemoteAddress()	returns the IP address of the server for the current socket
$setBlocking(pBlocking)	makes a socket blocking or non-blocking
$SetChunkSize(pChunkLength)	sets the chunk size

The oSocket object class has the following methods.

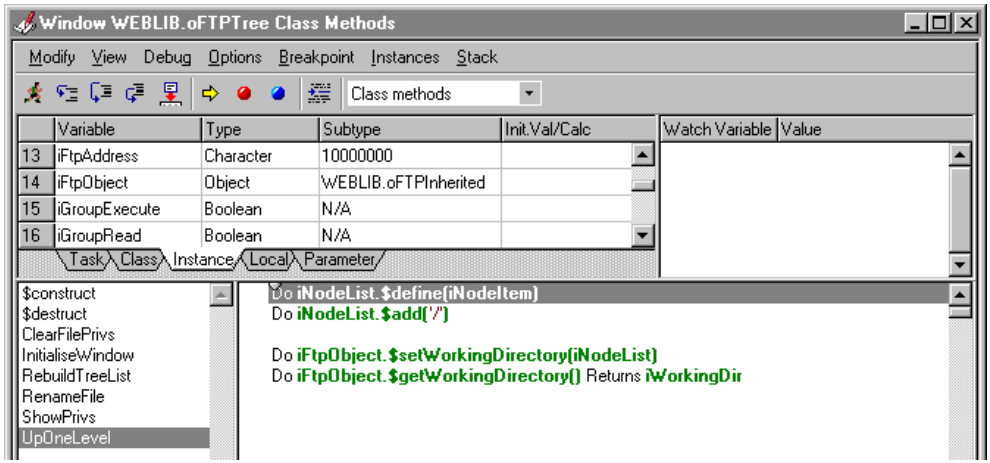
Method	Description
$send()	sends a message
$receive()	receives a message
$getSocketNumber()	returns the socket number
$getPortNumber()	returns the port number
$getLastError()	returns an error number
$getBlocking()	returns the Blocking status
$getSpyList()	returns a list of sockets
$setLogging(pLogging)	allows the switching of logging on/off

The oSpy object class maintains a list of sockets, and has the following methods.

Method	Description
$construct()	builds a list of sockets
$spySocket(pSocket,pOperation,pBuffer)	adds a socket to the list
$returnSpyList()	returns a list of sockets

Using the Net Object Classes

The best way to find out how to use the Net object classes is to look at the windows created via the Net window wizards. For example, take a look at the FTP window which uses the oFTP and oFTPInherited object classes. The FTP window contains an instance variable called iFTPObject based on the object class oFTPInherited, which is itself a subclass of the oFTP object class. You can use the methods in the object class or one of its superclasses. For example, the UpOneLevel method in the FTP window uses the $setWorkingDirectory and $getWorkingDirectory. Here is the code for the UpOneLevel method; note the iFTPObject variable.



Chapter 12—Accessing Your Database

The SQL Browser lets you access and maintain your proprietary databases such as Oracle, Sybase and Informix, in addition to accessing OMNIS data files using OMNIS SQL. This chapter provides an overview of how you connect to your server database and create a SQL form to view and update your data. It assumes you are familiar with standard relational database terminology, including ANSI SQL.

OMNIS Studio includes a separate Data Access Module (DAM) to connect to each server database supported in OMNIS. The DAMs are either specific to a particular DBMS or database technology, or to middleware that lets you connect to any of a large number of different databases and file servers. Specifically, OMNIS provides DAMs for direct connection to Informix, Oracle, Sybase, and DB2 Universal Database, and middleware DAMs for ODBC, and EDA.

OMNIS Studio provides three *SQL classes* that let you access your remote server database: these are *schema* classes, *query* classes, and *table* classes. The schema and query classes let you model the data structure of your server database, while table classes provide the interface and methods that let you manipulate your data via OMNIS list and row variables. You can define an OMNIS list based on any of the SQL class types and use the SQL methods to manipulate your server data via the list.

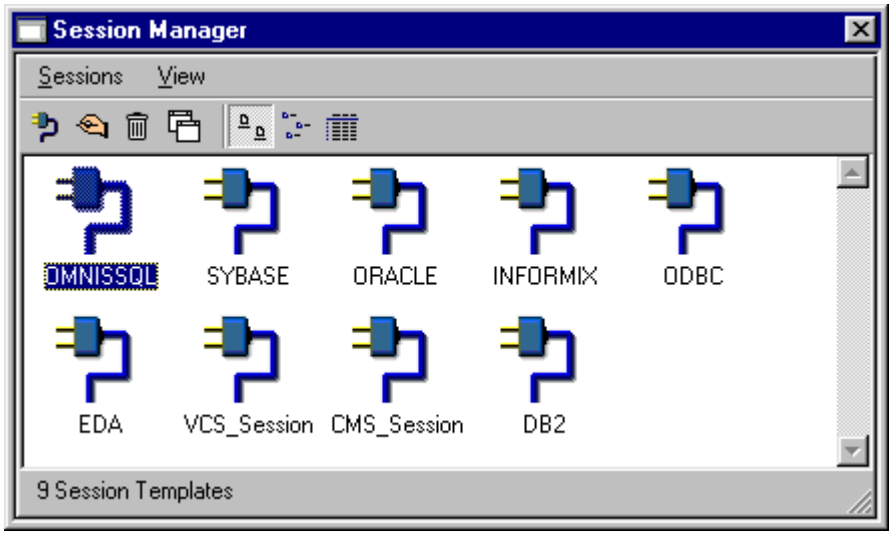
Connecting to Your Database

To connect to your server database in OMNIS Studio you need the appropriate client and networking software for your chosen server database and operating system, available from the various database and clientware vendors.

Having set up your networking and client software, you can start OMNIS and logon to your database using the SQL Browser. When you connect to your server database in OMNIS you create a *session*.

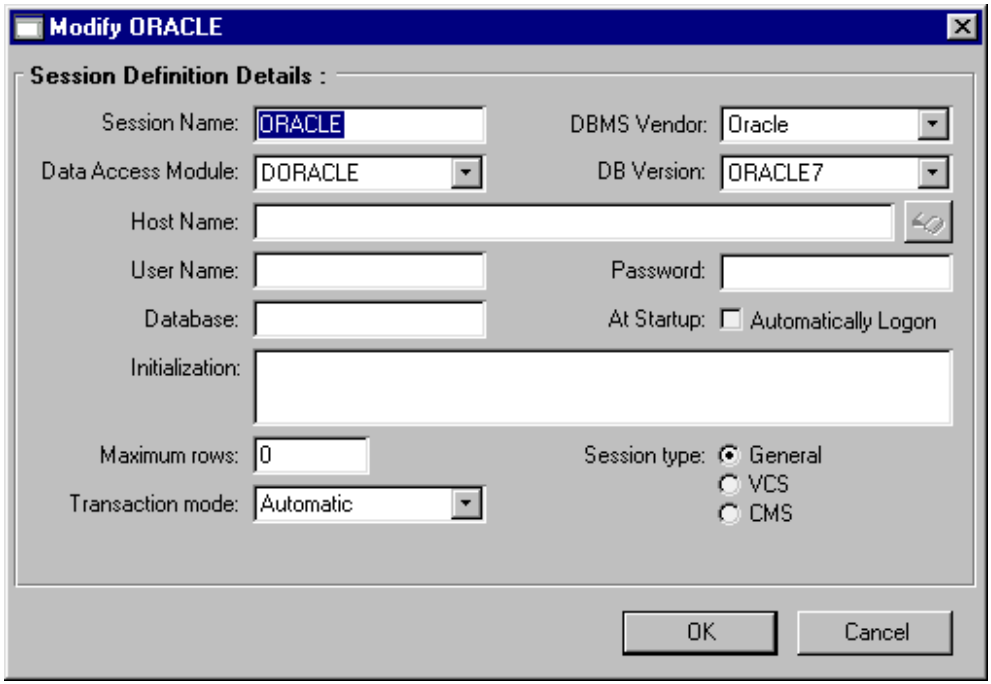
To access your database

- Start OMNIS and open the SQL Browser from the Tools menu; note that the SQL Browser is empty until you open a session
- In the SQL Browser, select Modify Session from the Session menu



Each icon in the Session Manager is a template that specifies the logon details for a particular database session. You can create a new session, or copy an existing one and modify it, but in most cases you can edit the templates provided.

- In the Session Manager double-click on the session appropriate to your server database



- In the Modify Session window edit the details for the session, including the database version, host name, database name, user name, and password

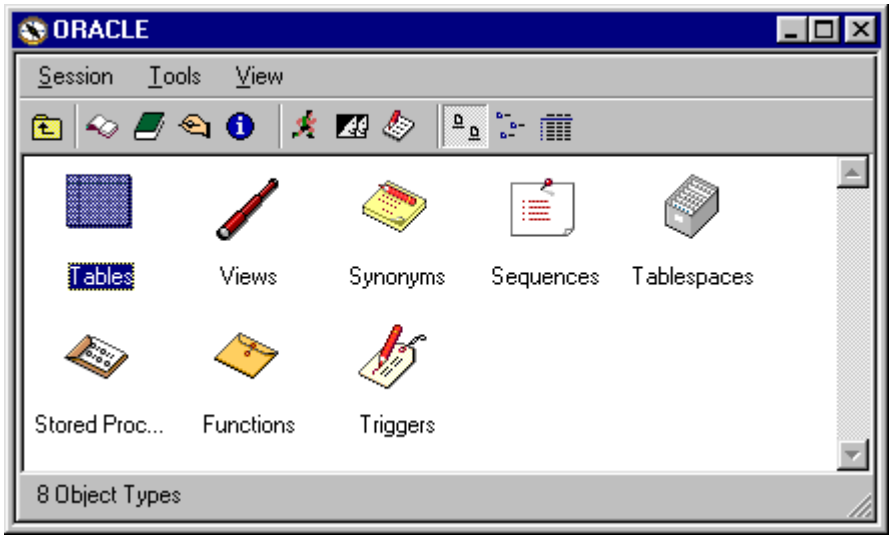
You may not need to supply all of this information for your particular database. For example, for ODBC and EDA you need to supply the database vendor's name and version, but this is not necessary for the direct connections.

- When your session details are complete click on OK
- Close the Session Manager
- In the SQL Browser, select Open from the Session menu and choose your session name from the submenu

To view the objects in your database

- Double-click on your open session in the SQL Browser

For example, if you access an Oracle database you may see something like the following.



To view the tables in your database

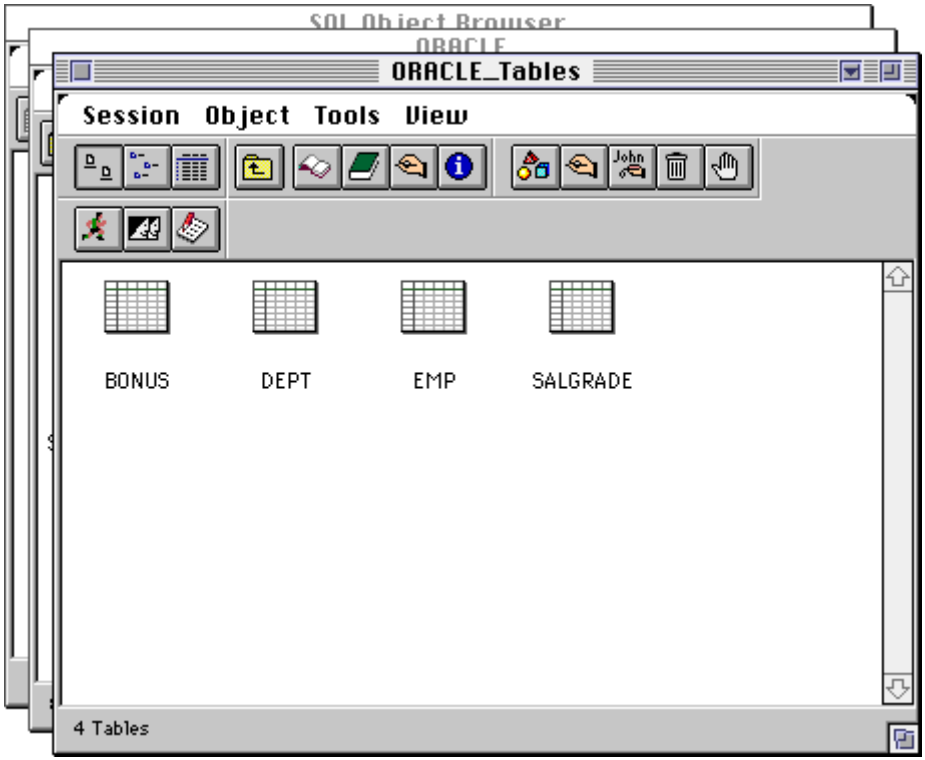
- Double-click on the Tables group in the SQL Browser

Enabling Your Client Application

Having logged on to your database using the SQL Browser, next you need to create the appropriate OMNIS classes in your library to enable data access between your server database and OMNIS client application. You can do this quickly and easily using drag and drop in the SQL Browser.

To enable your client application

- Open your library in the standard IDE Browser, or create a new one
- Assuming you are logged on to your database, display the tables in your database in the SQL Browser, as described in the previous section
- Drag the appropriate server table from the SQL Browser onto your library in the standard IDE Browser



OMNIS adds a schema class with the same name as your server table to your library, and displays it in the IDE Browser. OMNIS may also add a new table class which references the new schema class. This depends on a preference which you can set via the SQL Browser Tools>>Options menu item.

Creating SQL Forms

To view and edit the contents of your database you can create a window or SQL form in your client application, based on the schema class(es) you created using the SQL Browser. You can do this using the form wizards in the Component Store. Using the Window or Form wizards is described in more detail in the *Window Classes* chapter earlier in this manual.

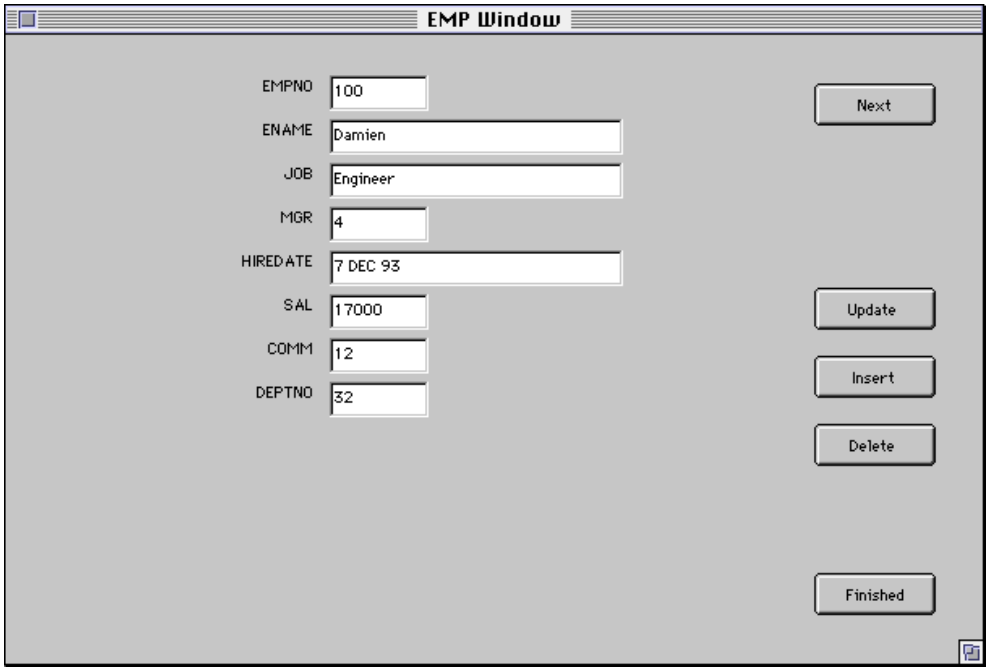
To create a SQL form

- Open the Component Store by pressing F3/Cmnd-3 and click on the Window Classes button in the toolbar
- Drag the **SQL Form Wizard** from the Component Store onto your library in the standard IDE Browser
- Name the new window class and press Return, or click in the Browser
- In the wizard dialog choose the type of form, either One field per column, Display Grid, Enterable Grid, or Parent/Child Window, then click on Next
- Select the database columns you want to include on your form, or check a table name to include all columns; for the Parent/Child window you need to select the parent and child columns
- When you have specified the columns for your form, click on the Next button
- Select a session, or leave all the sessions unchecked to use the default session, then click on Create

OMNIS creates and opens the new SQL form ready for you to modify or open to view your data.

To open your SQL form

- Press Ctrl/Cmnd-T, or Right-click on the window and select Open Window from the context menu



Depending on the type of form you choose you can insert data into your database using the Insert button, or you can next through your data using the Next button. You can modify this window or add further methods for SQL data access, which is covered in the *OMNIS Programming* manual.

Printing Database Information

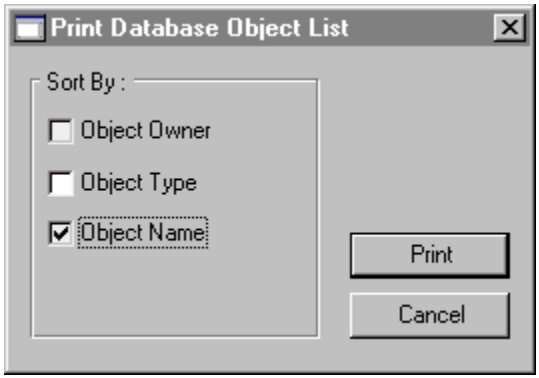
You can print information for a list of objects in your database. The information includes the object name, type, owner, and schema (where applicable for the database). Similarly, you can print groups of objects in your database, such as the Tables or Views group, as well as individual tables and views.

To print an object list for your database

- Click on the database and select Session>>Print Database in the SQL Browser menu bar

or

- Right-click on the database and select Print Database from its context menu



The criteria for the sort are shown in the dialog. For example, if you click on Type and Name, the object list is sorted on Type then by Name.

To print a group of objects in your database

- Open your database in the SQL Browser and display the object groups, such as Tables, Views, Synonyms, Stored Procedures
- Right-click on the object group and select Print Objects from its context menu

To print an object in your database

- Double-click on the object group, such as Tables or Views
- Right-click on the object and select Print Object from its context menu

Viewing and Inserting Data

The SQL Browser **Object** menu lets interact directly with your server database. You can view and insert data into the current table without creating any SQL classes or SQL forms in your library.

To view the data for a table

- Click on the table and select Show Data from the Object menu

or

- Right-click on the table and select Show Data from the context menu

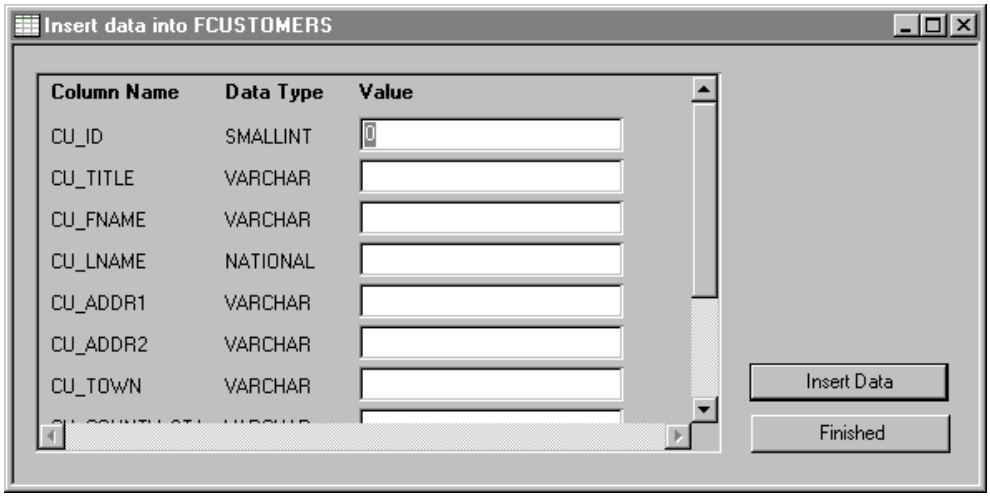
The Show Data option creates a Select statement for the current table and sends it to your server automatically via the Interactive SQL tool. The results of the Select are displayed in the lower pane of the ISQL tool.

To insert data for a table

- Click on the table and select Insert Data from the Object menu

or

- Right-click on the table and select Insert Data from the context menu



The SQL Browser creates a window containing an entry field for each column in your table. The window lets you insert a single row of data into your database.

Data Access Wizards

The Component Store contains a number of Wizards to help create database connections. The database wizards are under the Object Classes pane in the Component Store.

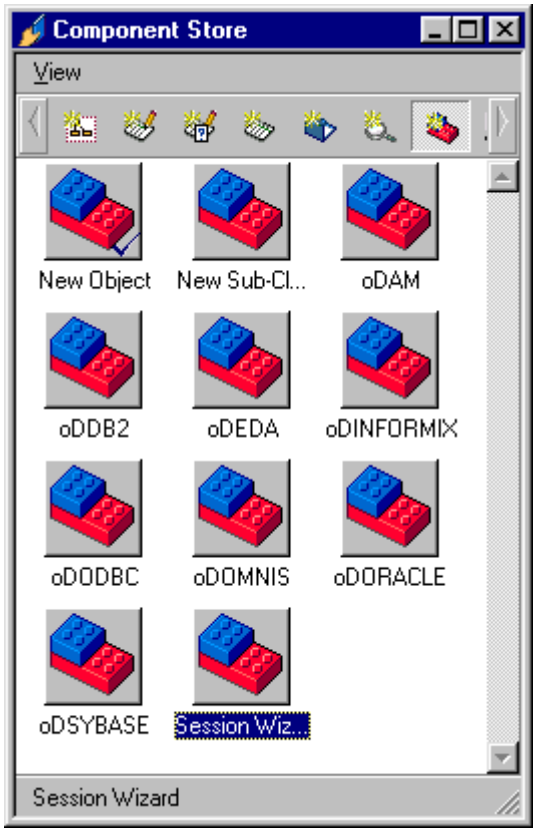
Session Wizard

The Session Wizard lets you create an object class based on a specified session in the SQL browser. The object class created in the wizard contains methods that let you logon and browse your database. The wizard also creates a logon window that provides an interface to the object class, and saves you having to create your own logon window. The wizard lists only those sessions that are correctly set up in the SQL Browser, that is, the wizard will not list any session templates to which you have not added full logon details such hostname and password.

To create a session object

- Open your library in the standard IDE Browser

- Open the Component Store by pressing F3/Cmnd-3
- Scroll the Component Store toolbar and click on the Object Classes button
- Drag the Session Wizard onto your library in the IDE Browser

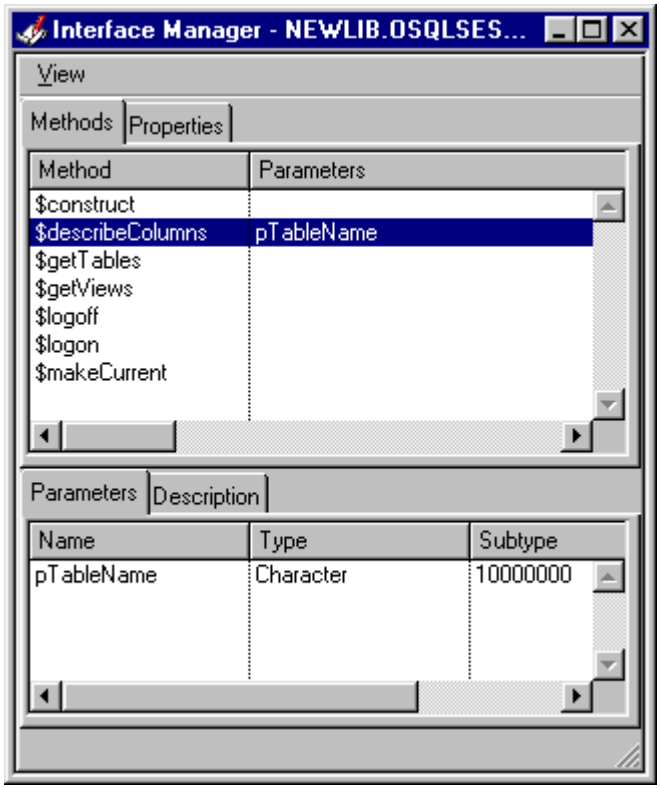


- Name the new object class and press Return, or click in the Browser
- Select a session from those available
- Check or uncheck the Create logon window button as appropriate
- Finally click on the Create button

The new object class is added to your library, together with the Logon window if you elected to create one, its name being your session name prefixed with the letter w. You can examine the object class in the Interface Manager.

To examine the session object

- Right-click on the object class and select Interface Manager from its context menu



To use the session object class in your application, for example in your own logon window, you need to create a variable based on the object class. You can call any of its methods using the syntax ObjectVarName.$methodname. Alternatively you may have created a logon window using the Session Wizard, in which case all the necessary code is written for you. You can either use the logon window as it is, or modify it for your purposes.

The logon window created using the Session Wizard contains Logon and Logoff buttons, and Username and Password entry fields. The second pane lets you browse the tables and views on your database.

You can examine the logon window in design mode to see how the object class is used. The window contains an instance variable called iSession which is based on the object class you created in the Session Wizard. To implement the Logon button, for example, you need only execute the method Do iSession.$logon() to log on to your database.

The object pane contains a set of wizards, one for each of the DAMs in OMNIS. These are all derived from a basic object class called oDAM, which is also in the Component Store, and provide the underlying functionality for you to create connections to your database. The Example Browser, accessed from the Tools>>Example Browser menu item, shows how you can use the data access wizards.

General Troubleshooting

This section gives you a few tips regarding general troubleshooting. For more detailed or server-specific hints and tips, refer to the *Server Specific Programming* chapter in the *OMNIS Programming* manual.

Many problems with logging on and accessing your database are caused by incorrect installation or configuration of either the server, network, or client software. See your DBMS and network installation instructions for guidelines and problem solving. If you are having trouble getting connected, you probably need the help and advice of your IS department and/or database administrator.

If possible, you (or your friendly database administrator) should try to connect to the server from your machine using another DBMS tool to make sure the client software, network connections, and server are all functioning, before you try to connect via OMNIS. Having established your server connection, logging on via OMNIS is easy!

In addition, you could check the following.

– Check the DAM is installed in the EXTERNAL folder under the main OMNIS folder. Furthermore, make sure there is a DAM for the server you want to access, and check it is available for the operating system you are currently using.

– Go back to the SQL Browser and check the logon parameters in your session template, including the name of the DAM (it should include the D prefix), also check the database version, host name, user name, and password

– If you get specific server or database errors, look them up in your DBMS manual; this will usually give you a clue about where to look for the problem

Chapter 13—Library Tools

OMNIS provides a number of tools for managing and maintaining libraries, and for changing the OMNIS design environment itself. For example, you can change the default classes available in the Component Store in OMNIS, you can add your own icons to OMNIS, and you can import and export data to and from your application. This chapter describes

- *Component Library*
  lets you edit the classes and templates in the Component Store
- *Welcome Library*
  lets you create a library which introduces your application at Startup
- *Icon Editor*
  lets you add your own icons to the OMNIS icon data file
- *Importing and Exporting data*
  lets you import data into an OMNIS list, or export OMNIS data to a file
- *Library tools*
  that let you check and retokenize your libraries, and make libraries private
- *Passwords and Security*
  lets you add passwords and security to your library
- *Localization*
  lets you translate your libraries and OMNIS itself into another language for deploying in international markets or enterprises

Component Library

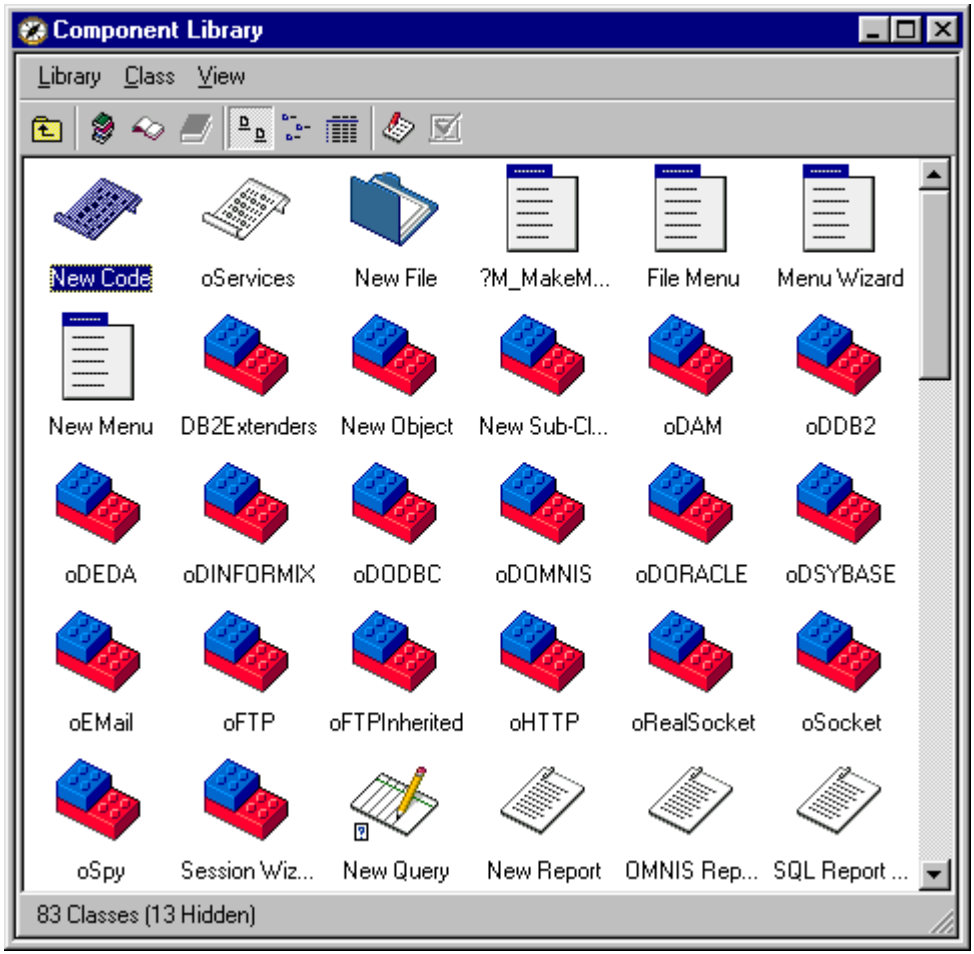
If you have started to create an application you will have used the Component Store already to create classes and other components. The Component Store contains class templates and wizards, window and report fields, graphics, and external components. This section describes how you can change the contents of the Component Store by modifying the Component Library, the OMNIS library that populates the Component Store.

To modify the Component Library you must open and close it from the Component Store itself. You *cannot* open it via the Browser as you would a normal library. Unless you want to copy classes from another library to the Component Library, it is recommended that you close all other libraries before you edit the Component Library. You should also make a backup copy of the Component Library before you change it. The Component Library is called COMPS.LBS and is found in the STUDIO folder under the main OMNIS folder.

If you have upgraded from an earlier version of OMNIS Studio and you have added your own components to the Component Library, you will need to add these components to the new Component Library supplied with the current version of OMNIS Studio. You will find a library called COMPCOPY.LBS in the CONVERT folder which lets you copy components from an old version of COMPS.LBS to a newer version. Save your old copy of the Component Library, somewhere other than in the Studio folder, open COMPCOPY.LBS, and follow the instructions. *Note that COMPCOPY.LBS will NOT copy changes you have made to the classes supplied by OMNIS Software in the Component Library. It only copies new classes that you have added to COMPS.LBS.*

To open the Component Library

- Open the Component Store, or bring it to the top by pressing F3/Cmnd-3
- Select View>>Show Component Library In Browser from the Component Store menu bar, or Right-click on the Component Store and select the same option



The contents of the Component Library is shown in the Browser. You can switch the Browser to Details view and click on one of the column headings to show the different types of component either by Name or Type. You can hide and show specific types of class in the Component Library using the Browser Options (press F7/Cmnd-7).

Modifying Classes and Fields

You can view and modify the classes in the Component Library as you would standard classes in your own libraries. You can also create classes in another library and copy them to the Component Library. The Component Library contains a number of default classes, either class templates, wizards or particular classes containing fields and other objects.

The names of some of the classes in the Component Library are used in the Component Store toolbar for the different groups of objects, while other class names have a special

function. Classes named “New <classtype>“ are the default templates for all new classes you create in your own libraries. Those that begin with an underscore “_” contain fields and other components for specific class types, such as window fields and report objects. Any other classes in the Component Library, such as the class called SQL Form Wizard, are the wizards and templates that appear under the different class buttons in the Component Store. Class names that begin with “?” are invisible in the Component Store and are used by the wizards.

Any changes you make to the classes in the Component Library are reflected in the Component Store when you close the Component Library.

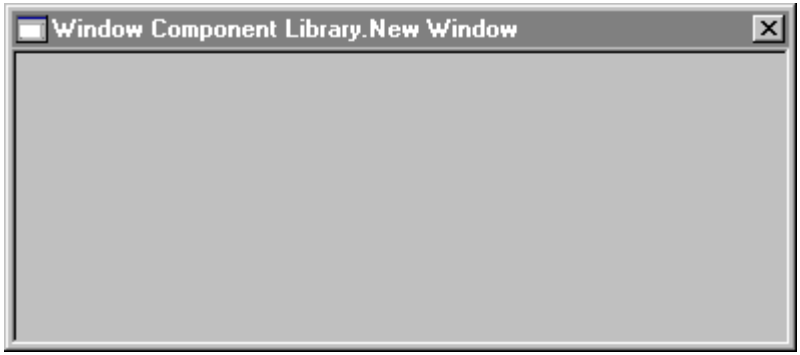
To modify a class in the Component Library

- Double-click on the class in the Browser

or

- Right-click on the class and select Modify from its context menu

For example, to modify the default window class, open the class called “New Window”.



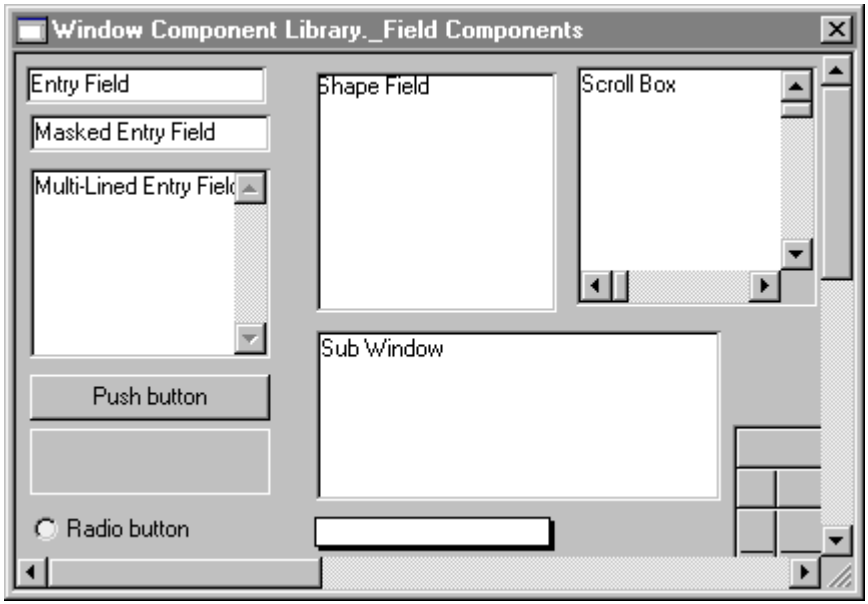
You can change the properties of this window class, or any of the other default classes. For example, you could change the size or background color of this window, or its window style. When you next create a new window from the Component Store it will have exactly the same properties as the New Window class in the Component Library.

When you have made your changes to a class

- Save the class from the File menu and close it

To modify a default window object in the Component Library

- Double-click on the window class called "_Field Components" in the Browser



The _Field Components class contains all the window fields available in the Component Store. You can change the properties of these fields to change the properties of the default fields that appear in the Component Store. Similarly, the window class called _Background Components contains all the background objects for window classes, such as lines and ovals. The _Report Components and _Toolbar Components classes contain the objects for their respective class types.

You can close the Component Library at any time, but you must close it from the Component Store itself; you cannot close it from the Browser. The Library>>Close option in the Browser remains grayed out while the Component Library is the current library.

To close the Component Library

- Bring the Component Store to the top by pressing F3/Cmnd-3
- Select the View>>Show Component Library In Browser option in the Component Store menu bar, that is, uncheck the option

Adding Classes and Fields

Rather than modifying the classes in the Component Library you can add your own, including classes that contain your own components based on the standard OMNIS field

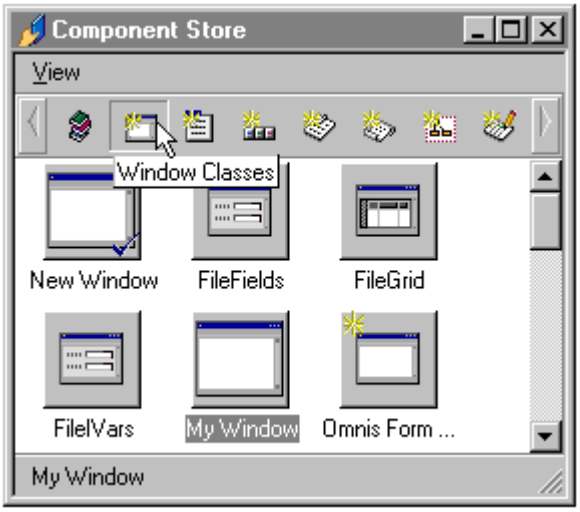
types. Your own classes in the Component Library can be called what you like, but a class containing fields or other window components must start with an underscore. You can also assign an icon to a class to appear in the Component Store. The names and icons of your own classes in the Component Library are used in the Component Store toolbar.

In addition to creating classes and changing their properties you can add your own methods to the objects in the Component Library. When you create objects from your own classes in the Component Store they will contain your own methods.

To add a class to the Component Library

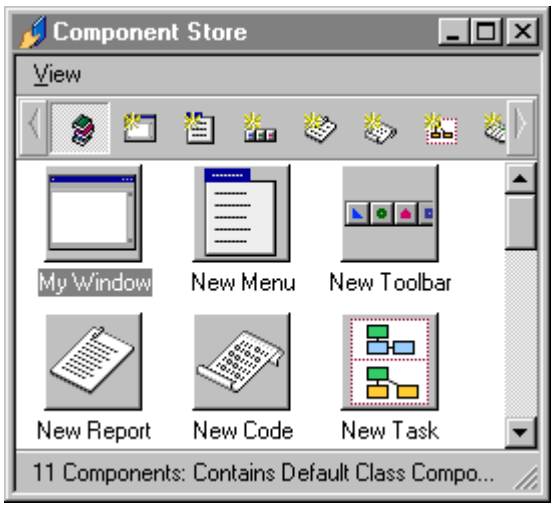
- Assuming the classes in the Component Library are visible in the Browser, select Class>>New>>[class] from the Browser menu bar
- Name the class and double-click on it to modify it
- Edit the properties of the class, including its **componenticon** property
- Save the class and close it

When you close the Component Library the class you created will appear under the appropriate group of classes in the Component Store toolbar. For example, if you add a window class called My Window Template to the Component Library it will appear under the Window Classes button in the Component Store.



Under each button in the Component Store toolbar you will see the default classes contained in the Component Library called New <Class> and marked with a check mark. You can make your own class the default by Right-clicking on the class and selecting Make Default. If for example you make a class called My Window Template the default window class, it

will appear under the Default Classes button in the Component Store toolbar and will be the basis for all new window classes you make from the Class>>New option in the Browser. When you create a class from the Browser or Component Store, an exact copy of the default class template is created in your library.



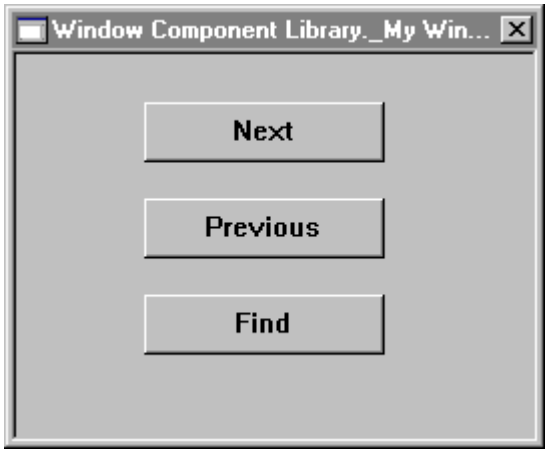
To add a class containing your own components

You can create windows, reports, and toolbar classes in the Component Library which contain your own window fields, report fields, and toolbar controls. You can add methods to these objects too.

- Select Class>>New>>Window/Report/Toolbar from the Browser menu bar
- Name the class including an underscore at the beginning; for example, create a window class called _My Window Objects
- Add fields or other components to the class and change their properties

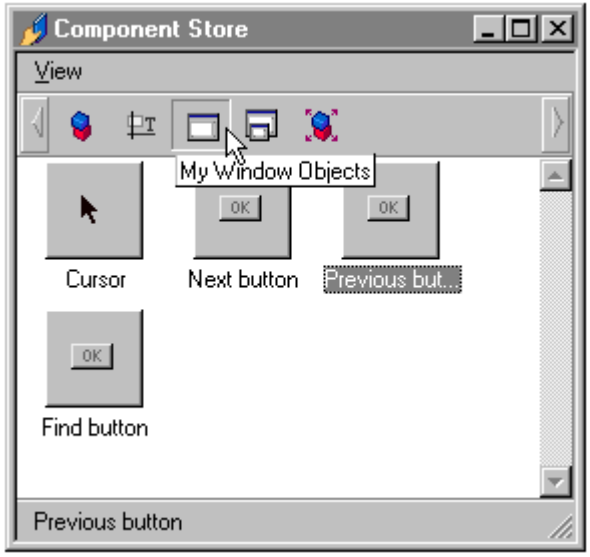
The **name** property of an object is used to label the component in the Component Store. You can change the **text** property to change the text that appears in the object; for pushbuttons, this is the text that appears on the button face. You can also set an object's **fieldstyle** property to add a style. Furthermore, you can add methods to your own components, which is particularly useful for pushbuttons and toolbar controls, and for adding event handlers to window objects.

For example, you could create your own set of database buttons, set the **buttonmode** and **text** properties for each button, and add your own methods.



- When you have finished modifying the objects, save the class and close it

When you close the Component Library the contents of the Component Store will update automatically showing any new classes or field components you have added. For example, the My Window Objects class described above is added to the Component Store toolbar and when selected displays the buttons added to the window.



Creating your own Wizards

As well as creating class templates, you can create your own wizards in the Component Library which are displayed in the Component Store along with the standard class templates and wizards.

Wizards can vary greatly in complexity, but usually consist of a template class and at least one wizard window that prompts the user for input. The main template class is the class that appears in the Component Store which the user drags onto their library. The template class contains some essential code in a method called $setup() and any other methods or objects you want to appear in the new class. The $setup() method must include parentheses in its name and can contain literally any code you want, but usually will initialize the wizard process and open any other wizard windows. Any other windows or classes you open or reference from the main template class must be named ?CLASSNAME to hide them in the Component Store.

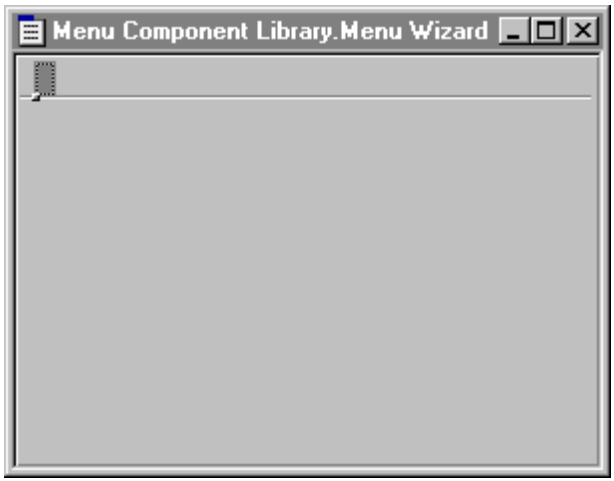
When the user creates a class using a wizard the following things happen.

1. The user drags the wizard or template class from the Component Store onto their library in the IDE Browser.
2. OMNIS highlights the name of the new class in the Browser prompting the user to enter a new name.
3. When the user enters a new name and hits Return, the Browser calls the $setup() method in the main template class.
4. The $setup() method initializes the wizard process and opens any wizard windows contained in the Component Library required for input from the user.
5. The wizard window contains code that steps through the different stages in the creation process and creates any objects or methods required in the final class.
6. Lastly the wizard deletes the $setup() method in the new class and opens it in design mode, ready for the user to edit or use it.

The best way to understand wizards is to look at one of them in the Component Library. For example, take a look at the Menu Wizard.

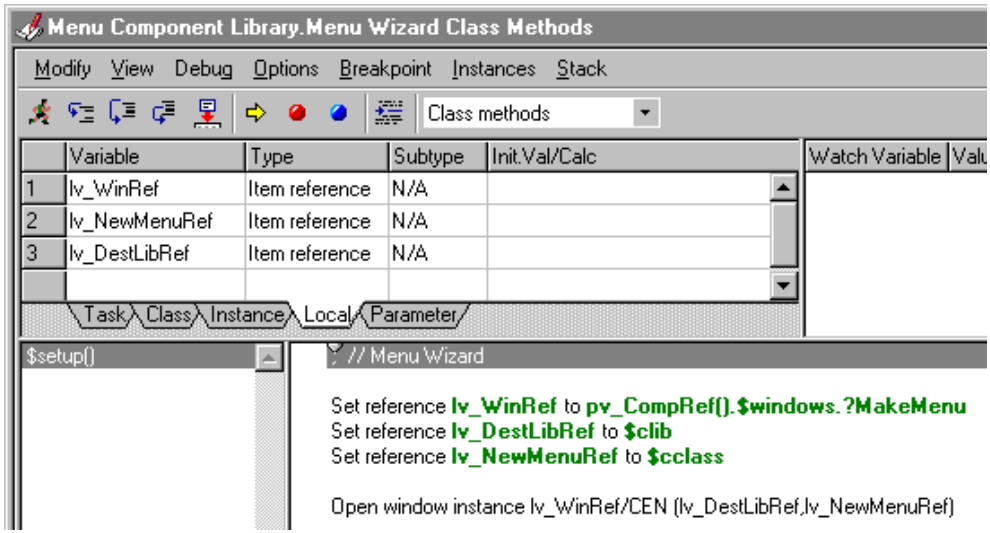
To examine the Menu Wizard in the Component Library

- Open the Component Library and display its contents in the Browser
- Open the menu class called “Menu Wizard”



This is the class that appears in the Component Store and the user drags onto their library and renames. The menu contains no menu lines or code except for the $setup() method, which is called after the new class is created and named in the Browser.

- Right-click on the menu and select Class Methods to view its methods



The $setup() method contains some essential code and variables to initialize the wizard. The method contains three local variables: lv_WinRef stores a reference to the wizard window, that is in this case, the window named ?MakeMenu; lv_DestLibRef stores a reference to the user's library which is $clib; and lv_NewMenuRef stores a reference to the new menu class created, which is $cclass when this method is called. The $setup() method also contains a parameter variable pv_CompRef which accepts a reference to the Component Library that OMNIS sends by default which is used to open the wizard window. These references are setup using the *Set reference* command.

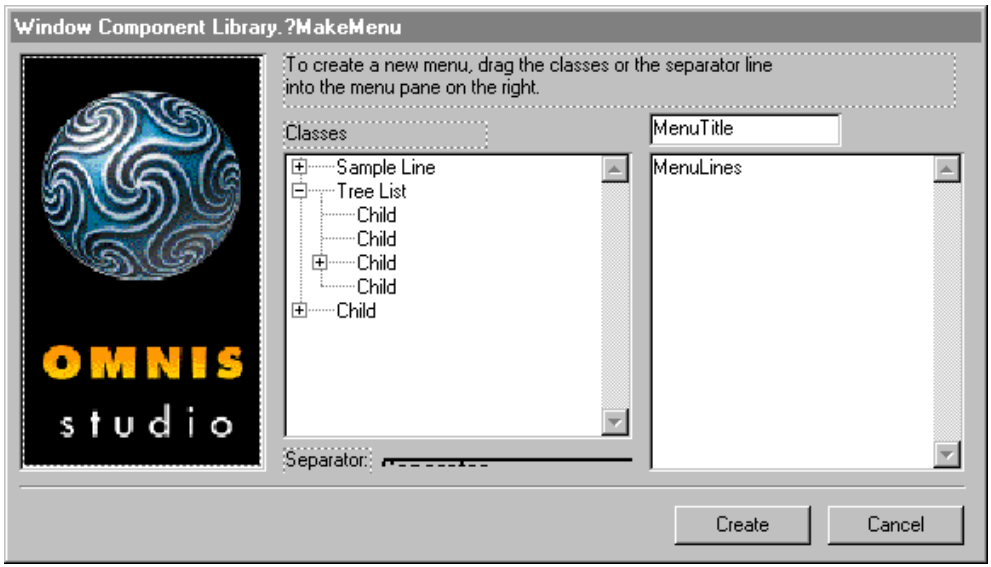
```
Set reference lv_WinRef to pv_CompRef().$windows.?MakeMenu
; the parens in the reference to the Comp Lib are required
Set reference lv_DestLibRef to $clib
Set reference lv_NewMenuRef to $cclass
```

The final piece of code in the $setup() method opens the wizard window stored in the Component Library.

```
Open window instance lv_WinRef/CEN (lv_DestLibRef,lv_NewMenuRef)
; alternatively, you could use the $open() method
Do lv_WinRef.$open('*',kWindowCenter,lv_DestLibRef,lv_NewMenuRef)
```

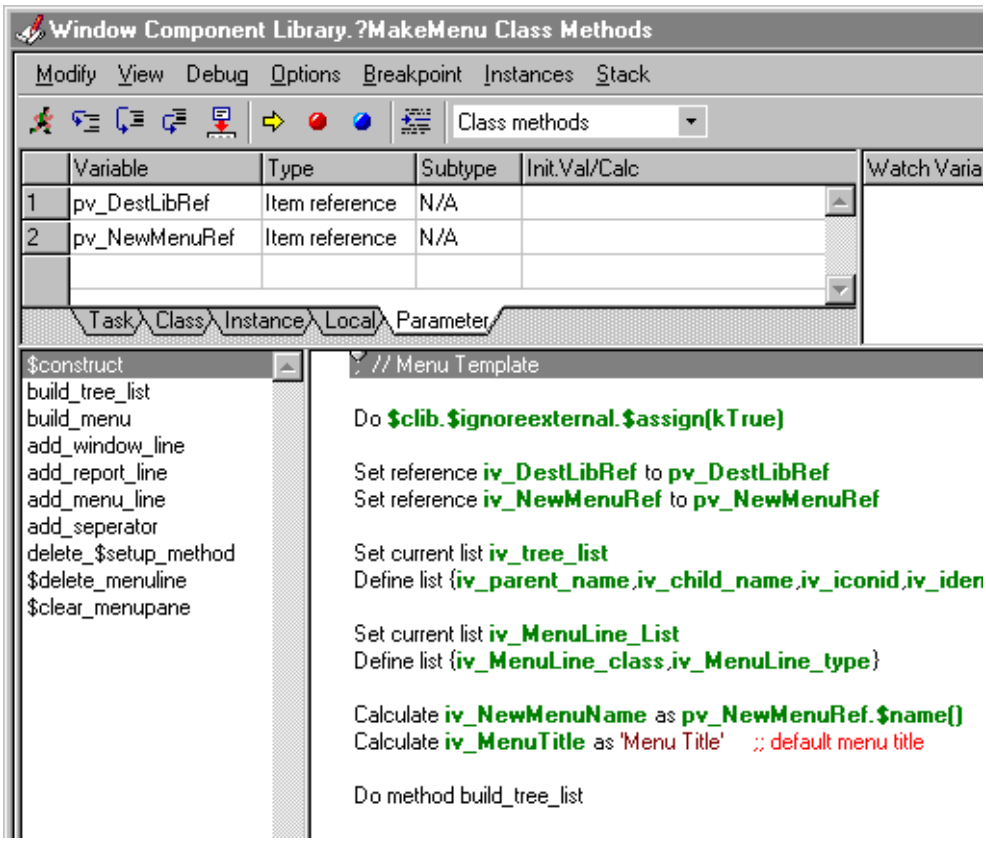
The *Open window instance* command or the $open() method pass references to the destination library and the new class, and open the window in the center of the screen. The '*' assigns a default instance name to the menu wizard window, which you don't need to know.

- Open the wizard window called ?MakeMenu



This wizard window contains many different fields and objects to allow the user to create a menu. The list on the left lists all the window, report, and menu classes in their library, and the list on the right lets the user construct their new menu. The wizard window also contains several methods that create the new menu class and build its own methods. Your own wizards may contain some or all of these elements. The other essential code you need to look at is in the class methods for the wizard window.

- Right-click on the wizard window and select Class Methods to view its methods



The $construct() method contains code that initializes the menu wizard, such as building the list of classes in the user's library and setting up the tree list that displays the list of classes. The $construct() method in your own wizards will be different, but should contain the line

```
Do $clib.$ignoreexternal.$assign(kTrue)
```

which allows access to the classes and methods in the Component Library. The $construct() method also contains two parameters to accept the values passed to it from the $open() method in the template class: pv_DestLibRef accepts a reference to the destination or user's

library, and pv_NewMenuRef accepts the name of the new menu class. You can use the *Set reference* command to assign these values to suitable instance variables.

```
Set reference iv_DestLibRef to pv_DestLibRef
Set reference iv_NewMenuRef to pv_NewMenuRef
```

The remainder of the code in the menu wizard is specific to this class, but the wizard contains one other essential method to remove the $setup() method from the new class. In the menu wizard, this is called from the method behind the Create button, as follows.

```
; $event() method for Create button on ?MakeMenu wizard
On evClick
   Do method build_menu           ;; builds the menu
   Do method delete_$setup_method (iv_NewMenuRef)
   Modify class {iv_NewMenuRef}   ;; opens the new menu
   Close window instance $cinst   ;; closes the wizard window
```

The method to remove the default $setup() can be called what you like, but here is the code

```
; code for delete_$setup_method
; contains parameter var pv_newmenu_ref to accept ref to new menu
; and local var lv_setupmthd_Ref to refer to the $setup() method
Set reference lv_setupmthd_Ref to
   pv_newmenu_ref.$methods.//$setup()//
Do pv_newmenu_ref.$methods.$remove(lv_setupmthd_Ref)
; removes the $setup() method from the new menu
```

The remainder of the Create button method opens the new menu class in design mode, using the *Modify class* command, and closes the menu wizard window.

Before creating your own wizards, you may like to look at some of the other wizards supplied in the Component Library. The SQL Form Wizard and OMNIS Form Wizard create windows of one sort or another and use subwindows to implement the various stages in the wizard process. The SQL Report and OMNIS Report classes use a similar set of wizard windows to create SQL and OMNIS reports. Open each of these classes and take a look at their $setup() methods to see how they work and what other classes they use.

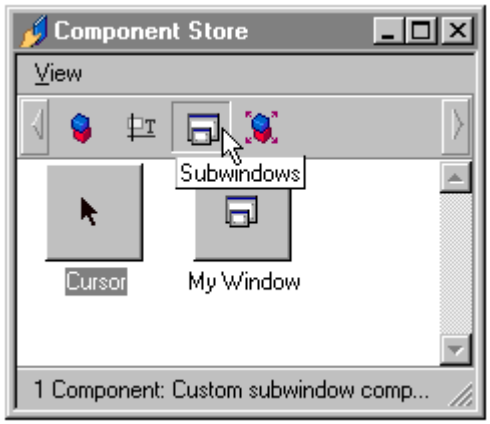
Showing Other Classes in the Component Store

Apart from adding your own templates and wizards or changing the default objects in the Component Library, you can display classes in your own libraries in the Component Store. Specifically you can show superclasses and subwindow classes in the Component Store.

You can show any class that supports inheritance in the Component Store by setting its **issupercomponent** property. All types of class except code, schema, file, and search classes support inheritance. You can specify the icon to appear in the Component Store for a superclass by setting its **componenticon** property. The superclass will appear under the appropriate group of classes in the Component Store toolbar, but you cannot make a

superclass the default class. If you create a class from a superclass in the Component Store, by dragging its icon on to your library in the Browser, the new class will be a subclass of the class shown in the Component Store.

You can also make any window class appear in the Component Store as a subwindow by setting its **issubwindow** property. When you are placing fields on a window, all subwindow fields appear under the Subwindows button in the Component Store toolbar.



The library containing your superclasses or subwindow classes must be open for them to appear in the Component Store, since the classes actually remain in your library and are not physically copied to the Component Library. If a library containing a superclass is not open inheritance will fail for all subclasses of the superclass, and similarly subwindow fields will fail if the subwindow class cannot be found.

Welcome Library

When you start OMNIS for the first time a library called “Welcome to OMNIS” is opened. This library introduces you to the product and lets you create a new library or open an existing one. You can create your own Welcome library to introduce your own application.

The Welcome library, called Welcome.lbs, is located in the STUDIO folder and is loaded automatically. You can create you own library, using the same name, to do literally anything when the user starts OMNIS. For example, if your application contains a suite of libraries you can use the Welcome library to present them to the user, or like the OMNIS one you can create a number of slides to introduce your product. Before you create your own, it’s worth taking a look at the Welcome library supplied with OMNIS. You can load it as you would a normal library, but since it has a Startup_Task you need to stop this running when you open the library.

To look at the Welcome library

- Select the Library>>Open option from the Browser menubar
- Under Windows hold down the Alt key, or under MacOS the Option key, and go to the STUDIO folder
- Open the Welcome.lbs library and view its classes in the Browser

Examine the Startup_Task and Welcome window classes, and in particular look at their Class methods. The code in the Welcome library uses the $welcome() method which is contained in $root.$modes and has the options kWelcomeNewLibrary, kWelcomeLastLibrary, and kWelcomeToggleStop. You can set the mode of the Welcome library using these constants. For example

```
Do $root.$modes.$welcome(kWelcomeNewLibrary)
; prompts the user to create a new library
Do $root.$modes.$welcome(kWelcomeLastLibrary) Returns iLibPath
; returns the path of the last library opened
Do $root.$modes.$welcome(kWelcomeToggleStop)
; stops the Welcome library from opening at startup
```

Icon Editor

The icon data file supplied with OMNIS, called OMNISPIC.DF1, contains the icons used throughout the OMNIS 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 use the icon ID to identify the icon. For example, when you place a pushbutton on a window you can set its **iconid** property to assign an icon to the button. When OMNIS displays the object it looks up this ID in the icon data file and displays the correct icon.

The icon editor lets you create your own icons for picture buttons, toolbar controls, tab pane controls, or any object that can have an icon or picture attached.

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 or your libraries. In addition, you can store your own icons in each separate library in the #ICONS system table. The advantage of using #ICONS is that it is stored with a library and, like any other class, you can copy it to another library or check it into the OMNIS VCS.

If you are converting a library from OMNIS 7 Version 3 and you have added icons to your old OMNISPIC.DF1 file, please refer to the later section on *Old OMNISPIC Data Files*.

Creating your own Icons

When OMNIS starts up it loads the USERPIC.DF1 data file automatically, along with the OMNISPIC.DF1 file. OMNIS will not recognize any other names. To create your own icons in OMNIS you use the icon editor and save your icons in the USERPIC.DF1 data file or #ICONS for a library.

To open the USERPIC data file

- Select the Tools>>Icon Editor option from the main OMNIS menu bar
- Select the File>>Open Icon File option on the Icon Editor menu bar
- Navigate to the Icons folder in the OMNIS main folder and open USERPIC.DF1

To open the #ICONS system table

- Show all the classes in your library in the Browser using Shift-Ctrl/Cmnd-A
- Double-click on the #ICONS system table

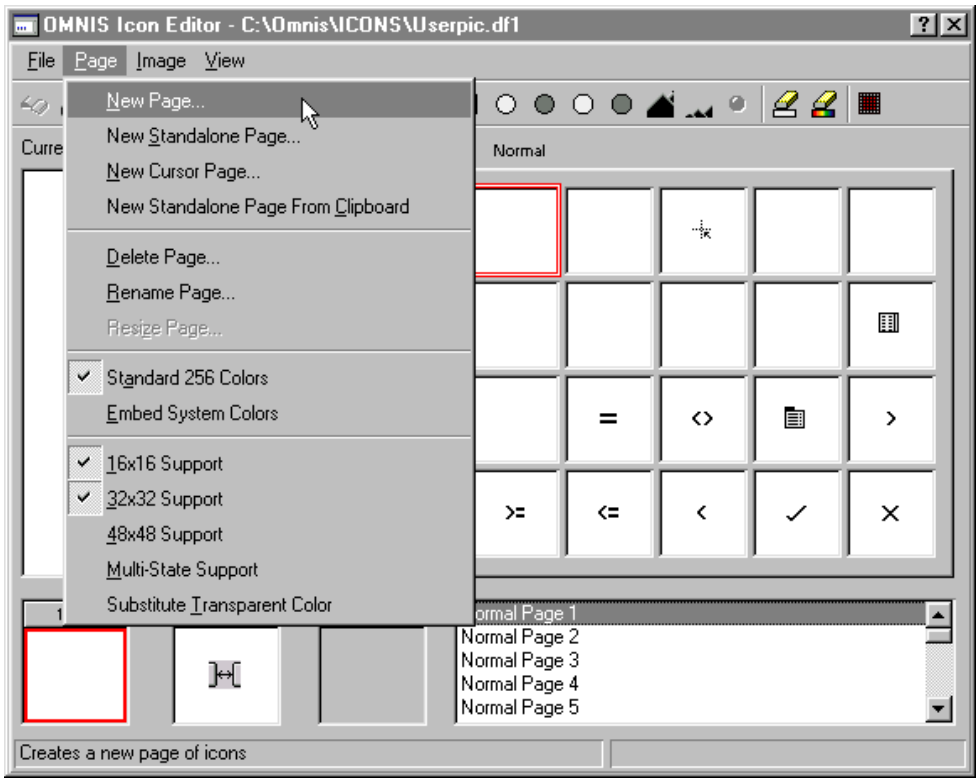
or if you have already opened USERPIC or #ICONS

- Select the USERPIC.DF1 or ‘Icons - Library name’ option from the Icon editor File menu

To add new icons to an icon data file or #ICONS, you have to first create a new page to store the icons.

To create a new page

- Select the Page>>New Page option in the Icon Editor



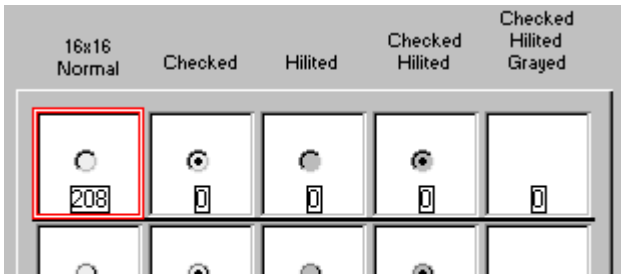
- Give the new page a name, something like “General 1”

The Icon Editor creates a new page in which you can create up to twenty different icons. You may want to create a separate page for each group of icons, for example, a group for general controls, a group for database icons, and another for PIM applications. You don’t have to fill a page before creating a new page, although each new page you use will result in more memory being required.

You can create a 16x16, 32x32, and 48x48 version for each icon by checking the Page>>16x16, 32x32, and/or 48x48 Support option in the icon editor. These support options apply for the whole page, so for example, if you create a 48x48 image for an icon and turn off 48x48 support for the page, all 48x48 images for the page will be lost.

The Icon Editor also supports full page icons or images that are larger than 48 x 48 pixels. In this case, each page supports a single image in which you can specify the image height and width as part of the page definition. You can modify the image in the same way as other icons, and assign an icon id to the page.

You can create an icon for the highlighted and checked state for controls such as check boxes and radio buttons using the Page>>Multi-State Support option. However, for multi-state icons you can create only four icons per page (one per row), with up to five different icons for different states or combinations of states. You only need to assign an ID to the Normal state for each icon, as shown.

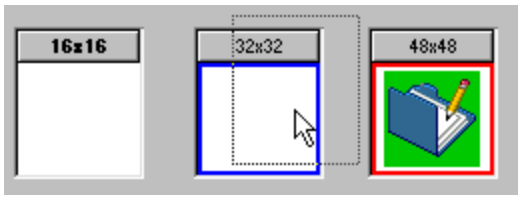


When the icon edit area has the focus, you can use the following keys to change the current tool in the editor. Note that no modifier keys are needed, just the letter key.

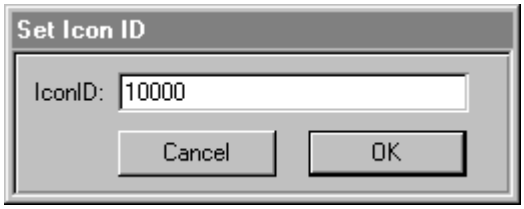
Key	Sets which tool?
m	marquee, or select
p	pencil
b	brush
i	eye dropper
k	bucket
e	eraser
+	increases eraser size
-	decreases eraser size

For most toolbar-type buttons you only need to create 16x16 versions of each icon. This is the most efficient approach, since if you limit the size to 16x16 for all the icons on one page, the page will be stored more efficiently and the USERPIC.DF1 data file will be smaller.

When you've created an icon in one cell, you can drag it into one of the other size cells and OMNIS will scale it to fit that resolution. It's better to create the icon at a larger resolution and scale it down. If you create all three views for an icon you can specify one particular view as the default using the Image>>Default to 16x16/32x32/48x48 option, or you can double-click on the button above the appropriate icon. The default size is shown on the button in bold. When you create a window object and select an icon for the object, it displays at the default size specified in the icon data file, unless you specify one of the other sizes.



When you have finished an icon and all its views you must give it a unique ID either using the Image>>Set Icon Id option on the Icon editor, or by double-clicking on an an icon in the grid. The Set ID dialog does not allow an ID that's already in use in the current icon data file.



Your own icon IDs must not be the same as the IDs in the OMNISPIC.DF1 file, therefore you should start at 10000 and allow a hundred numbers for each page or group of icons. For example, you could have a page of general icons numbered from 10000, then a page of database icons numbered from 10100, the next page would be numbered from 10200, and so on.

If the USERPIC.DF1 file is currently loaded in OMNIS, you can use any icons that you've added straight away. Similarly, if you change an icon that is currently in use on your screen you will see the changes immediately. However if you have created a new USERPIC.DF1, you must restart OMNIS to load the icon data file.

When you have created all your icons close the USERPIC.DF1 file, and close the Icon Editor.

Editing an Icon Data File

You can change the visual appearance of any icon, but you should avoid changing the ID of an existing icon. If you change the default size of an icon used in your application you may get unexpected results. In most cases, it is better to create a new icon with a new ID rather than radically change an existing icon.

Storing Icons in a Library

You can store icons on a per library basis in the system table class called #ICONS. If you can't see #ICONS in the Browser, use the Browser Options (F7/Cmnd-7) to show the System Tables. You can edit the #ICONS system table in the Icon Editor by double-clicking on the class in the Browser. The $iconid search order for any object is #ICONS, OMNISPIC, USERPIC.

Embedded System Colors

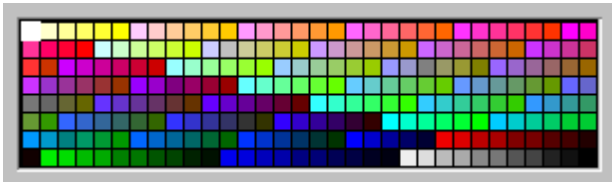
The OMNIS icon editor pages are always stored in 256 colors. The color palette is the standard OMNIS color palette. Under Windows, controls such as buttons and check boxes change color depending on what you have set in your Control Panel. As OMNIS simulates check boxes and radio buttons with icons from OMNISPIC.DF1, the bitmap needs to change color when the system colors are changed. To achieve this, you can embed system colors into icon pages.

The Page menu lets you switch between Standard 256 colors mode and Embedded system colors mode. Both modes appear to use the standard OMNIS color palette, but in Embedded system colors mode, color indexes 32, 64, 96, 128, and 160 are remapped to five system colors. You can use these system colors in your icons and OMNIS will use the colors currently loaded in the user's system. If the user changes their system colors while OMNIS is running, icons are reread from the appropriate icon data file and the correct colors are displayed.

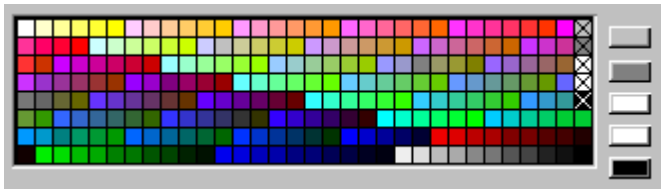
When you edit an image using system colors, the colors shown in the editor are not system colors, but the original color as seen in the standard OMNIS color palette (for example, dark red may be used instead of a system color). This helps OMNIS find colors to replace with system colors.

OMNIS supports five system colors: kColor3Dface, kColor3Dshadow, kColor3Dhilite, kColorWindow, kColorWindowText.

The standard OMNIS color palette is shown below. This palette is used for all icons in OMNISPIC and USERPIC files.

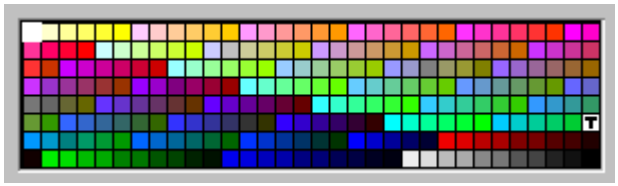


The standard OMNIS color palette with embedded system colors switched on is shown below. The system colors are shown in the right-hand column of the palette replacing the standard color indexes. You can also use the larger buttons to select a system color.



Transparent Color

In addition to system colors, you can embed a special color which represents a transparent area in your icons. Select the Page>>Substitute Transparent Color option to show the transparent color in the palette, shown with a T and currently set to bright green. Select the color and use it to paint or fill areas in your icon. Any pixels that use this color appear transparent in OMNIS.



Old OMNISPIC Data Files

The OMNISPIC.DF1 data file supplied with OMNIS 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 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 your picture buttons and other controls, assuming they are not larger than 48 x 48, which is the largest size supported. If you haven't changed the old OMNISPIC.DF1 file at all, you don't need to do anything, that is, you can 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 supplied USERPIC.DF1. Before you convert your old OMNISPIC.DF1 file, make sure you have a secure backup.

To convert your old Omnispic data file

- Quit OMNIS and remove or rename the USERPIC.DF1 file supplied with OMNIS in the Icons folder
- Restart OMNIS and open the Icon Editor from the Tools menu
- 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, save the converted file as USERPIC.DF1, and restart OMNIS

When you restart OMNIS the USERPIC.DF1 file will be loaded automatically. If you want to add icons to OMNIS you should add them to the USERPIC.DF1 data file, as above.

Custom Cursors

You can design your own cursors for fields and store them in either the USERPIC.DF1 data file or the #ICONS system table in your library. You create the cursors in the Icon Editor in very much the same way as for custom icons. Each cursor can contain four images to accommodate the differences in size and color required for cursors under the different operating systems.

To create your own cursors

- Open the Icon Editor from the Tools menu
- Select the File>>Open Icon File option from the editor menubar
- Navigate to the Icons folder under the main OMNIS folder, and select the USERPIC.DF1 icon data file

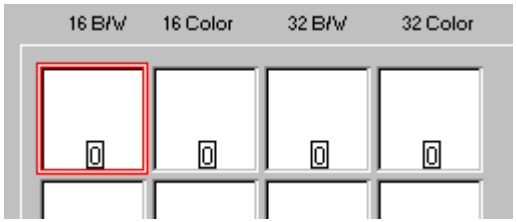
or to store cursors in your library

- Show the system tables in the Browser and double-click on #ICONS

for an icon data file or #ICONS

- Select New Cursor Page from the Page Menu, enter a Page name and press OK

Each cursor page allows up to four different cursors: for each cursor you should add the equivalent image for each platform.



– 16 B/W
  MacOS in black & white mode; only use black & white (plus transparent color if set)
– 16 Color
  MacOS in color mode, 16x16 pixels in size
– 32 B/W
  Win16 mode; only use black & white (plus transparent color if set)
– 32 Color
  Win32 mode, 32x32 pixels in size

- Draw an image for each platform
- Double-click on one of the cursor images to set the ID

Specifying the cursor IDs is the same as for icons, that is, you should choose an ID over 100000, to avoid conflicts with OMNIS icons, up to a maximum value of 16777215.

Cursor Hot Spot

The hot spot for a cursor is the single pixel on screen that is activated when you click the mouse, such as the tip of a pencil tool. You can specify a hot spot for a cursor in the Icon Editor by clicking in the image with the Hot Spot tool.

- When you are done, select File>>Save from the Icon Editor menubar and close the Userpic file

Cursor Properties of Objects

You can set the cursor properties of an object in the Property Manager or using the notation.

To set the cursor properties for an object

- Click on the object and open the Property Manager by pressing F6/Cmnd-6
- Click on the Appearance tab and set the **cursor** property
- Click on the Action tab and set the **nodropcursor** and **candropcursor** cursor properties

In the Property Manager, the cursor, nodropcursor, and candropcursor properties have two parts: an object can have either a standard or custom

– **Standard cursor**
  a dropdown list which lets you select one of a number of standard cursors represented by OMNIS constants (which are not stored in an OMNIS icon data file or #ICONS); the list contains the value kcursDefault, which you select if you want the default cursor for the object

– **Custom cursor** (specified by iconid)
  a dropdown list button which lets you enter or select the id for a cursor; the id identifies a custom cursor in an OMNIS icon data file or #ICONS; once you have entered a cursor id, the dropdown field in the first part of the cursor property is set to kcursCustom

You can use the notation to assign a cursor property, as follows:

```
Do $cobj.$cursor.$assign(kcursArrow) ;; assigns a standard cursor
Do $cobj.$cursor.$assign(100001)     ;; assigns a custom cursor
```

Importing and Exporting Data

You can import and export data to and from your OMNIS application using a number of different data formats. You may want to use the import/export facility prior to mail merging, creating spreadsheets, or for moving data between an OMNIS data file and a SQL database. You can import data from a file, port, or client.

You can import and export data using list or row variables. If you are going to use the data in conjunction with an OMNIS database, you must write methods to extract the data from the list and save it to the database or vice versa.

Data Formats

You can import from or export to virtually any application or database providing it supports one of the following OMNIS data formats

– Delimited (commas)
  a text format that separates data with tabs

– Delimited (tabs)
  a text format that separates data with commas

– Delimited (User delimiter)
  a text format that separates data with a user-defined character

– One field per line
  a text format that puts the data in a row on separate lines

– OMNIS data transfer
  a proprietary format that lets you transfer data between OMNIS databases

Null values are exported as empty if the OMNIS preference $exportnullsasempty is set.

Delimited (commas)

Each field is separated by a comma and each record or row of data starts on a new line. The newline character is ASCII 10 followed by ASCII 13 on a PC, but is just ASCII 13 under MacOS. You can enclose text fields in double quotes so that commas within the text data are not confused with field delimiters: this is set by the library preference **exportedquotes** or the *Enclose exported text in quotes* command.

Long character fields are exported, but control characters are replaced by spaces.

Delimited (tabs)

This file format is similar to delimited (commas) format, except that tabs (ASCII 9) are used between fields instead of commas. Each record or row of data starts on a new line. You can enclose text fields in double quotes so that tabs within the text data are not confused with field delimiters; this is set by the library preference **exportedquotes** or the *Enclose exported text in quotes* command.

Long character fields are exported, but control characters are replaced by spaces.

Delimited (User delimiter)

This format is similar to delimited (commas), but uses the character in the library preference $userexportdelimiter. This property takes a single character and defaults to ; (semi-colon). The kEXuser constant represents the user-defined character.

One field per line

With this format each field starts on a new line. OMNIS does not embed record information into the file. When you use this format to transfer data from one OMNIS application to another, the number of fields in the file must be exactly the same as the number of fields into which the data is stored, otherwise you will not get the expected results. If there are more fields in an export file than you want to import, specify some variables for the unwanted data so that the extra data is imported only as far as RAM and never stored in the data file.

If you have any doubts about the fields stored in the export file, load the file into a text editor and check it first.

Long character fields are exported, but control characters are replaced by spaces.

OMNIS data transfer

A proprietary format that lets you export data of all types from an OMNIS database, including pictures, long character fields and character fields that include control characters.

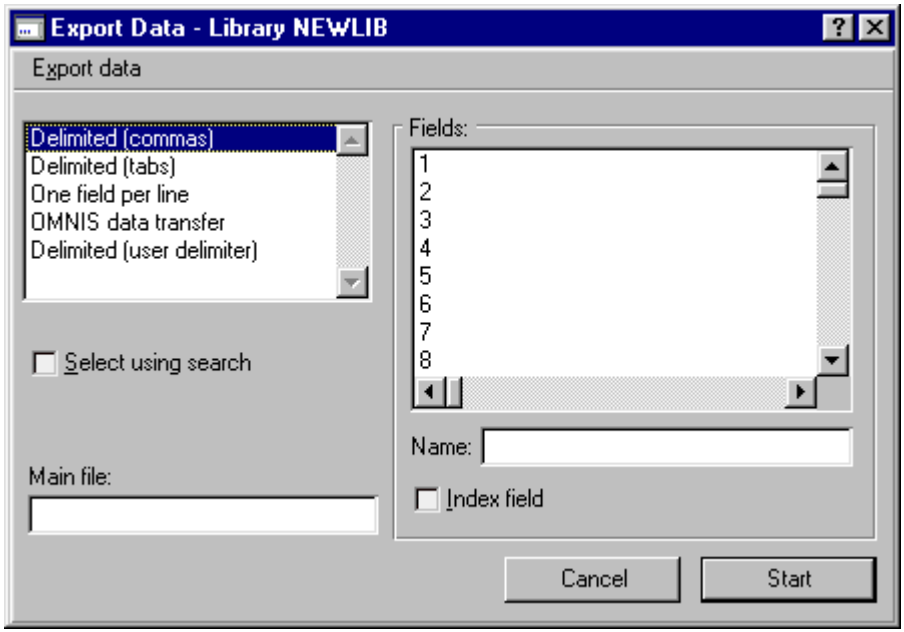
If an OMNIS data file should become corrupt because of bad media or similar hardware faults, you may be able to export data in this format and import it into a new data file.

Exporting Data

Using the Tools menu you can export data from an OMNIS data file.

To export data using the Tools menu

- Select the Tools>>Export Data menu item



- Select the required export format in the left-hand list
- Click in the Select using search box if you have defined a search class to filter out certain records
- Enter the main file for your data file in the Main File entry field

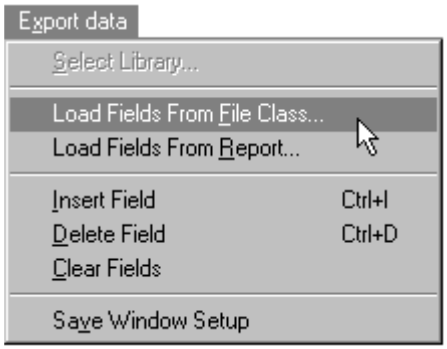
Next you need to construct a list of fields to be exported. The fields can be from the specified main file and from any connected files. To do this either

- Select a line in the Fields list and enter a field name in the Name entry field, or enter the field names from the Catalog

or

- Select the Load Fields From File Class option in the Export Data menu on the export data dialog

The Export data menu lets you insert, delete, and clear fields from the field list.



– **Insert Field**
inserts an empty line in the field list at the selected line; all field names in the list after the selected position are moved down

– **Delete Field**
deletes the field name in the list at the selected line; all field names in the list after the selected position are moved up

– **Clear Fields**
clears all field names from the field list

- Select a field that you want to index and click on the Index field check box

- When your list of fields is complete, click on Start

You are presented with a standard dialog into which you enter the export file name. Under Windows, the file name extension should be .TXT.

- Enter a file name and click on OK

OMNIS reads the data file in the order specified by the selected index field and writes the fields to the selected file. You can check the export file by opening it in a text editor.

The following sample data containing names and phone numbers was exported using the Delimited (commas) data format: the ID (indexed field), Title, FirstName, LastName, and Tel fields were selected from the database. Note that each value is enclosed in quotes.

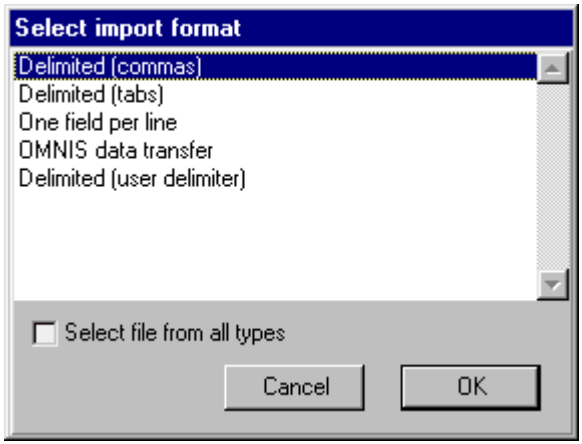
```
1,"Mr","James","Briggs","01326 671888"
2,"Miss","Thelma","Hodges","415 286 5378"
3,"Mrs","Lavinia","Williams","01394 243847"
4,"Dr","Peter","Eagleburger","514 538 4556"
5,"Rev","Leonard","Eccles","01452 974722"
```

Importing Data

Using the Tools menu you can import data into an OMNIS data file from an import file. The import file must be in one of the data formats described above. You can also import data into a list or row variable, which is described in the next section.

To import data using the Tools menu

- Select the Import Data option from the Tools menu

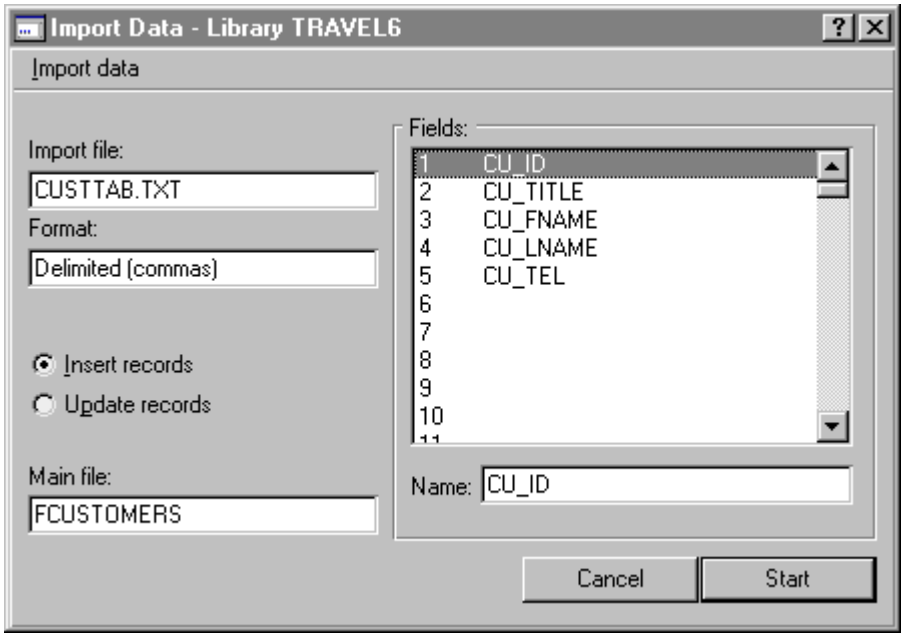


- Select an import data format and click OK

You are presented with a standard dialog in which you select the file to be imported.

- Select a file name from the list and click OK

Once you select a file name, the Import data window is displayed. It includes the Import data menu which lets you insert, delete and clear fields in the list.



The data in the export file will be read into the OMNIS fields in the order in which it is stored in the file. It is, therefore, essential that the field list you specify corresponds with that of the incoming data. If a line in the list is left blank, the data for that line will not be imported, and if there are too many fields in the list, the extra ones are left empty.

You need to construct a list of fields to be imported, although in some cases OMNIS will enter a list of fields for you. To add to the field list

- Select a line in the Fields list and enter a field name in the Name entry field, or enter the field names from the Catalog

or

- Select the Load Fields From File Class option in the Import Data menu on the import data dialog

- Select either the Insert records or the Update records option by clicking on the appropriate radio button

If you choose the **Insert records** option, new records are created from the imported data. The **Update records** option updates the existing records in your data file.

- When your field list is complete, select Start

The number of records imported is displayed when the import is complete.

For each record read from the file, a matching record is sought in the OMNIS data file. The first field in each record of the imported file is used to carry out an exact match find on the data file and the matching record updated with the incoming values. Records for which there are no matching records in the data file are ignored.

Importing Data into List and Row Variables

You can import data into a list or row variable using the *Import data* command. You can import either multiple rows or a single row depending on whether you specify a list or row variable in the *Import data* command. You must specify the list or row variable with the corresponding columns and column data types, although data conversion is performed when necessary. The list or row is cleared before import begins. When importing into a list, you can use its $linemax property to limit the number of rows imported. You can test the flag to see if there is more data to be imported, particularly if you are importing into a row variable on a record-by-record basis.

You can initiate the import process using *Prepare for import from file* which sets up the data format, and terminate it using *End Import*. For example

```
; importing into a list variable
Prompt for import file  ;; set up the file ..
Prepare for import from file {Delimited (commas)} ;; and format
Calculate List1.$linemax as 100
Import data {List1}  ;; brings in the first 100 rows of data
End import
Close import file
; process the data in List1

; importing into a row variable
Prompt for import file
Prepare for import from file {Delimited (tabs)}
Import data {Row1}  ;; get first row
While flag true
   ; process the data in Row1
   Import data {row1}  ;; get next row
End While
End import
Close import file
```

You can also import data one field at a time from file or port into the CRB, using *Import field from file* and *Import field from port*. These commands also have the option to import a character at a time if, for example, you need to identify non-standard delimiters (see below).

Exporting Data from List and Row Variables

You can export data either from a list or row variable to a file or port. The commands *Prepare for export to file* and *Prepare for export to port* specify an export data format and start the export operation. The *Export data* command exports all the data in a list or row variable. *End export* terminates the operation. For example

```
; exporting from a list variable
Prompt for print and export file
Prepare for export to file {Delimited (commas)} ;; set export format
Export data {List1}
End export
Close print and export file

; exporting from a row variable
Prompt for print and export file
Prepare for export to file {Delimited (commas)} ;; set format
Repeat
   ; load some data into Row1, clear the flag if finished
   If flag true
      Export data {Row1}
   End if
Until flag false
End export
Close print and export file
```

Prepare for export to file and *Prepare for export to port* both generate an error if no file or port has been selected.

Export data takes a row or list variable as a parameter and exports all rows from the list or a single row. There is an error if no *Prepare for export* command has been executed. The flag is set if the export is successful and cleared if it fails; this often denotes an ‘out of disk space’ condition. The first time *Export data* is encountered any heading to the export file required by the export format is generated from the list or row definition. You must use the same list or row definition when exporting to the same file, otherwise an invalid export file will result: this is particularly important if you use the One field per line data format.

End export terminates any export in progress and, if appropriate for the export format, adds a trailer to the export file. You must use *End export* before the file or port is closed otherwise the exported data may have an invalid format.

Exporting Data from a Report

Before you can export data from a report, you must create a report class containing all the fields to be exported. To make a report export its data you need to set its $exportformat property.

To create a report for exporting data

- Create a new report class from the Component Store, or from the Browser
- Add the fields from the Component Store
- Set the $exportformat property of the report to one of the export formats
- To print from a file class, set the $mainfile property to the main file in your database
- To print from a list, set the $mainlist property
- Save the report class and close it

You can print the report from the Browser, or from a method.

To print a report from the Browser

- Select the File>>Report Destination option from the main OMNIS menu bar, and set the destination to File
- Right-click on the report class and choose Print Report from the context menu

The resulting report contains data in the One field per line format.

To print a report from a method

If you print the report to either a port or a file, the fields defined in the report are exported in the selected export format. You can also export to the clipboard where you can examine and paste the data into other applications. For example

```
Set main file {FPorders}
Set report name R_Export
Send to file   ;; sets the print destination
Prompt for print or export file
If flag true
   Print report
   Close print or export file
End If
```

If the print file is not closed, further exported data is appended to the end of the file.

Translate Input/Output

The Windows environment uses the ANSI character set and not the standard DOS extended ASCII characters as used by many other programs and printers. When you export data the Translate input/output option forces OMNIS to convert ANSI to ASCII, and when you import data to convert ASCII to ANSI. This option only applies to ASCII/ANSI values greater than 127. The Translate input/output option is controlled by the $translateoutput root property. You can set it using the notation, or in the Property Manager under the Tools>>Options menu item.

Build export format list

You can use the *Build export format list* command to create a list of data formats and use a number between 1 and $linecount to select an appropriate data format from the list. This is particularly useful in non-English language versions of OMNIS where the data formats may have been translated. For example

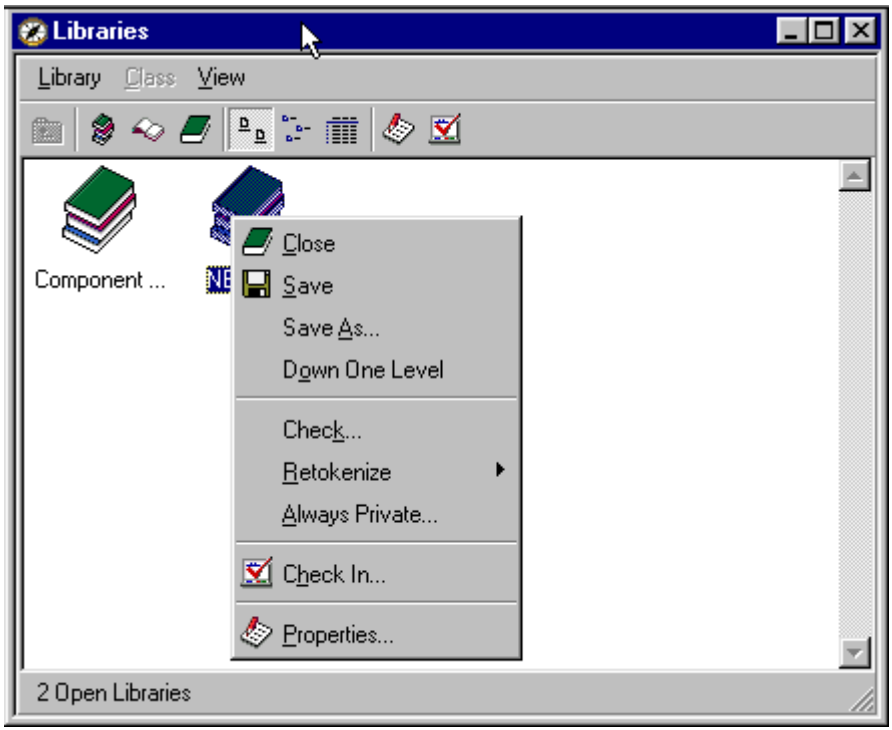
```
; This is language dependent
Set export format {Delimited (tabs)}

; this is universal
Set current list LOCAL_LIST
Build export format list (Clear list)
Set export format {[LOCAL_LIST(1,2)]}
; line 2 is tab-delimited in all language versions
```

Checking Libraries

You can check a library for disk errors or other inconsistencies using the Check option, which you can access by right-clicking on your library in the Browser.. Note that for a large library, this operation may take some time and to ensure data integrity, must not be interrupted once it has started.

To view the library tools context menu

- Right-click on your library in the Browser



To check a library

- Right-click on your library in the Browser
- Select the Check option in the library context menu

OMNIS closes all the design windows and scans all the classes in your library. On completion it reports any problems such as inconsistencies in indexes.

Retokenizing Libraries

As you write OMNIS methods, the method editor converts the commands you enter into a special internal form that replaces field and file class names with numbers. This process is called tokenization. The file class and field names are tokenized into their identifying numbers, the file class identifier and the field number within the file class. Tokenization occurs in all the libraries in your application.

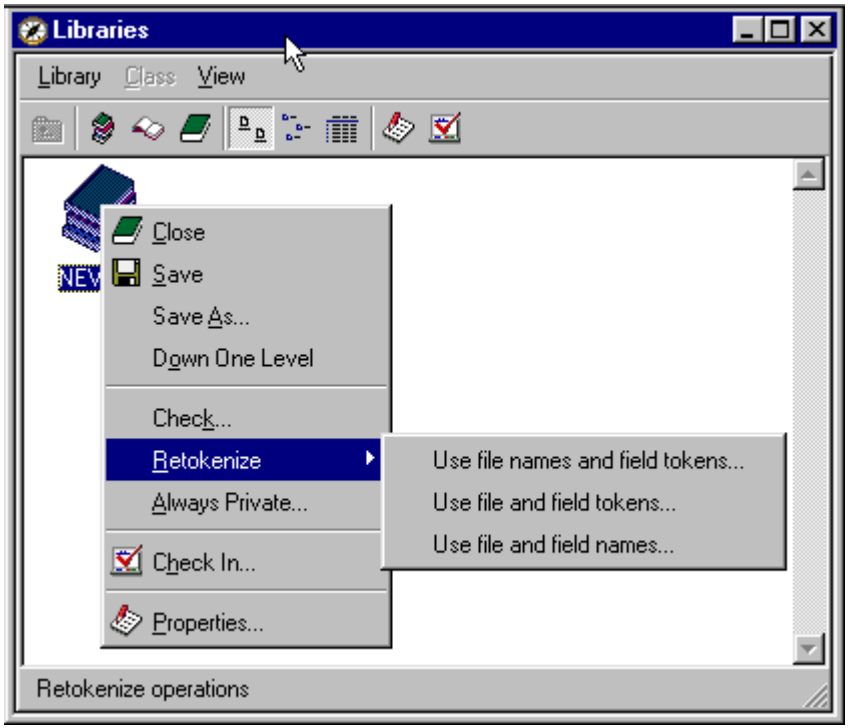
When you make a reference to an object defined in another library, you refer to the library name, the class name, and the field name. In this case, OMNIS tokenizes the field name only, leaving the file class and library names in their text form.

Retokenization ensures that all references to other libraries in the current library are correct. OMNIS searches the current library for external field references with the library specified and replaces the reference with a specific token.

WARNING Whenever you retokenize the current library, you must open all the external libraries to which the current library refers. If you don't references to external field names will be lost.

To retokenize a library

- Right-click on your library in the Browser
- Select the Retokenize option from the library context menu



The **Use file names and field tokens** option represents field references with the file class name and field number and ensures existing tokens are safe to use; this is the default behavior. You use this to convert references after completely tokenizing them with the next option below.

The **Use file and field tokens** option represents field references with fully tokenized file class and field numbers: this makes the library smaller and more efficient but you should use it only when the libraries are stable, just before deployment for example.

The **Use file and field names** option represents field references with file class and field names and no tokenization at all.

For the best mix of performance and safety, you should convert your libraries with **Use file and field tokens** just before deploying your application, then retokenize with **Use file names and field tokens** to continue working on the libraries for the next release. You should not continue working on libraries once you have fully tokenized the external field references.

Note that retokenizing specifies the object for conversion but does not change the standard format of OMNIS: any external field references you add after retokenizing are represented by the file class name and field number.

Private Libraries

If you make a library private its classes cannot be viewed and its methods are not accessible from other libraries. *Making a library private is irreversible.* You must save a copy of your library before you make it private in case you do need to modify it at a later date.

To make a library private

- Right-click on your library in the Browser
- Select the Always Private option in the library context menu

You can make a library private on a temporary basis by setting its **isprivate** property to true. In this case, you have to set this property each time you open your library; you could do this in the startup code in your library.

Locking Classes

You can lock classes on an individual basis. A locked class can no longer be viewed, deleted, changed, renamed, duplicated, or printed. *Locking a class is irreversible.* Therefore before you lock a class, make sure you have a copy.

To lock a class

- Right-click on the class in the Browser
- Select the Lock option in the context menu

You can hide an instance on a temporary basis by setting its **isprivate** property to true. You have to set this property each time the instance is opened.

Passwords and Security

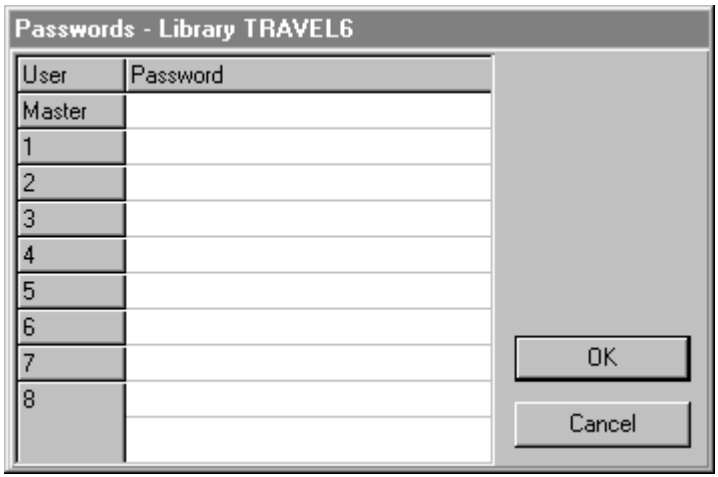
You can add up to eight user passwords plus a master password to each library. The passwords in your library are stored in the #PASSWORDS system table. You can use passwords to control access to the library itself, and to different objects in your library including menus and toolbars. Passwords are not hierarchical in the sense that user 1 has greater access than user 2, rather you grant particular access privileges to each separate user number. New libraries do not contain any passwords, therefore as master user you can define them, and assign user numbers to the objects in your library.

Once you have entered a master password, your library will prompt you for a password when you open it. The master user has unrestricted access to all parts of your library. When you open a library you can enter the master password or one of the other user passwords and OMNIS will grant you the appropriate access. The number of the current user(s) is stored in the $userlevel property for a library.

Setting Up Passwords

To set up passwords for your library

- Open your library in the Browser and show the system tables; you can press Shift-Ctrl/Cmnd-A to show all the classes in your library
- In the Browser, double-click on the #PASSWORDS system table



The Passwords dialog is empty for new libraries.

- Enter a password for the Master user, and for users 1 to 8

Passwords can be up to fifteen characters long. You should enter passwords for all users since leaving any passwords blank gives full access to a user who enters no password.

- Click on OK to close the window and register your passwords

From now on, you can edit the passwords system table only if you are the Master user, that is, if you open the library with the master password. It is therefore very important not to forget the master password!

Restricting Access to Menus and Toolbars

You can restrict access to menus, menu lines, toolbars, and toolbar controls by removing user numbers from these objects. These objects have the property $users, set by default to *12345678* to give access to all users. To restrict access to an object, delete the appropriate number from the string. For example, to restrict access to an object for user 3, delete 3 from the string, leaving 1245678.

When you restrict access to an object, the object is grayed out and users cannot select it. Setting the $users property for menus and menu lines is described in more detail in the *Menu Classes* chapter.

Controlling Access using Methods

You can test the $userlevel property to find out the current user. For example, the following method checks the current user number before allowing a data file record to be deleted.

```
; Delete method
If $clib.$userlevel = 3
   OK message {Your are not allowed to delete data}
   Quit method
End If
Delete with confirmation {Are you sure you want to delete?}
```

Alternatively, you can use a switch statement to branch according to the current user.

```
Switch $clib.$userlevel
   Case 1,2,3,4
      ; code for user 1, 2, 3, or 4
   Case 5,6
      ; code for user 5 or 6
   Case 7,8
      ; code for user 7 or 8
   Default
      ; code for the Master user only
End Switch
```

Having setup passwords in your library, OMNIS will always prompt you for a password when you open the library. However you can open a password prompt at any other time using the *Change user password* command.

Multi-library Projects

OMNIS lets you structure your application into one or more libraries that you can load either together or separately. This lets you

– break up your application into smaller, less complex subsystems

– develop the subsystems in different programming groups or departments

– test the subsystems or modules separately

– reuse libraries in different applications, mixing and matching reusable code without modification

Although OMNIS always ensures the integrity of objects, there is no built-in locking or concurrency checking to prevent two users from modifying the same object. If more than one user opens an object in design mode, the last one to close the object overwrites the changes made by the other users. There is no way to ensure that changes made to an object are seen by other users before the library is reopened: objects are cached in memory and it is not possible to predict when OMNIS will discard an object from the cache.

The OMNIS VCS provides you with a full-featured version control system for your OMNIS libraries and other components. If you put your application under version control, you eliminate the inherent risks involved in group development.

Localizing your Application

If you are developing OMNIS applications for an international market, you may want to translate your libraries and/or OMNIS itself into another language. OMNIS provides tools for localization in the LOCAL folder under the main OMNIS folder.

To translate a library you

– Export the text and literal strings in your library that require translation

– Translate the text

– Import the text back into your library

To open the translation library

- In the standard IDE Browser, select Library>>Open to bring up the standard open library dialog

- Locate the translation library called trans.lbs in the Local folder under the main OMNIS folder and open it

The translation browser appears. It displays any libraries you currently have open, with the exception of the translation library itself. If the library you wish to translate is not open, use the Library>>Open option on the translation browser menu to open it.

Exporting the Text in your Library

To translate your library you need to export all the text and literal strings in your library that require translation.

To export the text in your library

- Select Edit>>Export from the translation browser menubar

An export dialog appears; you can choose whether to export to an OMNIS data file or to a text file.

In either case, the Include Comments checkbox lets you filter out comments when you export text; generally you will not need to translate comments as they are not visible to end-users of your library.

Exporting to an OMNIS Data File

This opens a standard file dialog that lets you supply the name of the data file. Having named the OMNIS data file, the export process starts, and an export log keeps you informed of progress. You can export more than one library to a data file if you wish.

Exporting to a Text File

This opens a standard file dialog that lets you supply the name of the export file; the default is export.txt. OMNIS generates a file consisting of all the literal text strings as a standard ASCII text file in the format of the current platform. An export log keeps you informed of progress.

Translating the Export Data File

You edit the contents of the exported OMNIS data file in the translation library itself.

To translate the contents of the exported OMNIS data file

- From the translation browser menu, select Edit>>Editor

You are prompted for the data file name.

- Select the library you wish to translate from the list that appears when you select File>>Open

A grid field displays information about the translatable data. You enter changes in the New Value column. Alternatively, you can select Find and Replace from the Edit menu to look for specific strings; note that Find and Replace will not find partial strings. For example, if you have a test label whose old value is “My current label”, a search for the word “current” will not locate this label.

- Once you have completed your changes, use File>>Save to save them back to the data file

Translating the Export Text File

You can edit the contents of the exported text file in any standard text editor. If you use a Word Processor, you must ensure that you use it in text mode to avoid any special formatting characters being included in the text.

OMNIS structures the export text file as a sequence of three elements per string literal in your library. Each element is on a separate line in the export file. Each line starts with a keyword that identifies its purpose, as follows

– **REF=**
the location of the string in your library: *Do not change this*.

– **OLD=**
the original text string in quotes, including any control characters. *Do not change this*.

– **NEW=**
an empty string (“”) for you to fill in with the translated text

OMNIS regards all other lines as comments and ignores them when you import.

If an original string contains a double quote character, OMNIS places it within single quotes, and the NEW string should match it exactly. If any item contains more than a single literal, the OLD= line will consist of the right number of separate, quoted strings, each separated by a space. Strings can be up to 255 characters in length.

If you leave the NEW= empty, the OLD= string will be unchanged and untranslated. If you remove the quotes after NEW=, OMNIS replaces the OLD string with an empty string.

Importing Translated Text

Having translated the text in your library in either an OMNIS data file or text file, you can import it back into your library using the translation library.

To import translated text back into your library

- Select Edit>>Import from the translation browser
- Check the appropriate box to indicate whether you are importing from a data file or a text file and locate the file when prompted to do so

An import log and progress bar keeps you up to date with the import process.

- When you have finished, close the import log and the translation browser

Localizing OMNIS

Developers and distributors in non-English speaking countries may need to localize OMNIS. You can localize the following OMNIS internal items:

– The names of the days of the week

– The names of the months of the year

– Separator characters

– The text for Yes/No, OK/Cancel, True/False, Am/Pm and On/Off

– The national sort ordering

Storage of Localization Data

All libraries share the same set of data, stored in an OMNIS data file or *localization database* called OMNISLOC.DF1, located in the OMNIS local folder.

OMNISLOC contains a data slot for configuration data; each record in that slot contains a complete set of data corresponding to a particular language. It also contains a data slot with a single record, which identifies the current language, that is, the current set of configuration data.

The Localization Data

The following items are stored for each language.

Days of Week

This comprises 2 strings for each of the 7 days of the week, allowing for a full name such as Wednesday, and an abbreviated name such as Wed.

Months of Year

This comprises 2 strings for each of the 12 months of the year, allowing for a full name such as August, and an abbreviated name such as Aug.

Separators

These comprise the following:

- The decimal point used for all numeric fields
- The thousands separator used for numbers
- The function parameter separator
- The decimal point used when importing data
- The field separator used when importing
- The sequence used for quoting names in the notation

Standard Text Strings

These comprise the strings for Yes and No, OK and Cancel, True and False, Am and Pm, and On and Off.

National Sort Ordering

This is a text string which defines the sort ordering for national fields. To eliminate any cross-platform ambiguity introduced by the different character sets involved, this is stored as a 512 byte ASCII representation of the 256 byte sort ordering. Each byte in the sort ordering represents the position of the character equal to that byte in the sort order.

The natcmp() function

The natcmp() function lets you compare two values using the national sort ordering.

```
natcmp (value1, value2)
```

OMNIS converts both values to strings before doing the comparison.

OMNIS uses the same rules for comparing the strings as it does for normal strings, except that it performs the comparison using the national sort ordering.

natcmp() returns 0 if the strings are equal, 1 if value1 > value2, and -1 if value 1 < value2.

User Interface

The OMNIS preferences accessed from the IDE Tools>>Options menu line let you assign a new language from the dropdown list in the **newlanguage** property. The current language is shown in the **language** property. The language must already be defined in the localization data file.

The new language does not apply until you quit and restart OMNIS. Note that if the localization database is shared by several users, then the new language setting affects each of them, as soon as they restart.

An OMNIS library, OMNISLOC.LBS is provided that lets you create and edit language information. To use it:

- Take a backup of the OMNISLOC.DF1; you may prefer to work on the backup copy rather than the live copy, in which case you should make a working copy as well as a backup copy
- Open the OMNISLOC.LBS library, found in the OMNIS local folder. You are prompted for the location of the localization data file and a localization menu is installed on the IDE menu bar, to the right of the Tools menu
- Select Current Language to display the language in use
- Select Language Records to create a new set of language information, or to edit an existing one. This displays a dialog containing a set of tabbed panes and the standard OMNIS Insert, Edit, Find, Next and Previous buttons

You use the Next and Previous buttons to move through the records in the data file, the Find button to locate a particular language record, and Edit to modify data already present in the data file.

Two Insert buttons are available. Insert lets you create a brand new record, while Insert CV lets you make a copy of an existing language record and edit that. This is particularly useful for cases where there are only minimal differences between two language records. To use Insert CV:

- Display the language record you want to copy
- Click on Insert CV

A new record is created. Remember to edit the language name as well as the specific internal data.

- When all the data is input, click on OK to store it and close the library
- If you were working on a copy of the data file, move it back to the local folder

- Close the OMNISLOC library

Any fields that are left blank will default to a single space. Some of the fields on the General tab are limited in terms of which characters can be used; for example trying to define a letter as a decimal separator is not allowed, and will generate an error message.

Notation

There is no requirement to manipulate localization data at runtime, so the localization notation is minimal.

– **$root.$prefs.$language**
a read-only property which returns the name of the language OMNIS is currently using

– **$root.$prefs.$newlanguage**
a property that lets you assign the name of the new language, that is, the language OMNIS will use when it restarts

– **$hascurrlangnationalsortorder**
a property of a data file, for example
$root.$datas.DataFile.$hascurrlangnationalsortorder
if true the sort order matches that for the current language, and false otherwise

Every data file stores its national sort order. When you create a new data file, OMNIS stores the national sort order for the current language in the data file.

$hascurrlangnationalsortorder is assignable, but you cannot set it to kFalse only kTrue. When set to kTrue OMNIS drops all of the indexes from the data file, changes the sort order to that for the current language, and rebuilds all of the indexes.

Chapter 14—Version Control

This chapter describes how you use the OMNIS Version Control System, or VCS, to control OMNIS application development in a team environment.

An OMNIS application may consist of a number of elements. Apart from the OMNIS libraries and data files, you may have your own externals, components, text files and so on that are all necessary to the running of the application. In this chapter, the term *component* is used to refer to all these types of object and files stored on disk. Specifically, a *non-OMNIS component* is any disk file or external component other than an OMNIS class.

Version control lets you revise OMNIS library files and other application components systematically. In a team environment, with several people working simultaneously on the same application, you need to ensure that only one person can change a particular component at a time. Using the OMNIS VCS you can control the development of your OMNIS applications, or any other project involving many different files such as Internet or Intranet applications. Specifically, the VCS can manage OMNIS libraries or their classes, external components, DLLs or Code Fragments, OMNIS data files, text or word processor files, or any other types of file required in your OMNIS application.

To place OMNIS libraries under version control you check them into the OMNIS VCS from the Browser in OMNIS Studio, or for non-OMNIS components from the File Browser in the VCS itself. All the components you check into the VCS are kept in a *project*. The VCS stores each project in its own *repository* database, which can be a server database or an OMNIS data file.

The OMNIS VCS provides all the functionality to set up, manage, and use version control, including

- Setting up a project and repository database
- Checking in OMNIS libraries and other non-OMNIS components
- Managing and supervising users
- Building projects and libraries for distribution
- Managing projects and granting user privileges
- Setting VCS options

Compatibility with OMNIS 7

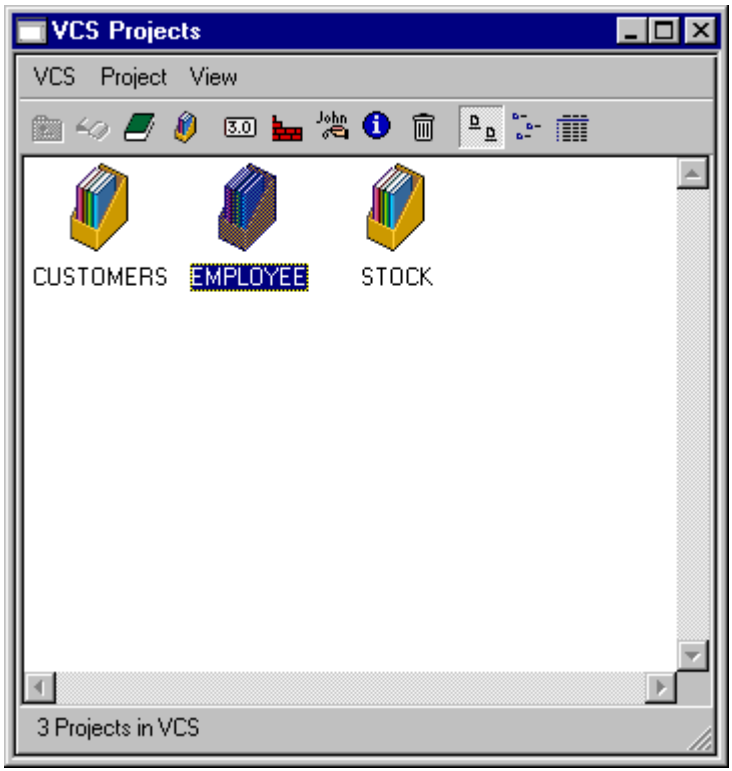
You cannot access a repository set up in OMNIS 7 in the OMNIS Studio version of the OMNIS VCS, nor can you use an OMNIS Studio VCS repository in the OMNIS 7 VCS. However, you can have an old OMNIS 7 VCS repository and a new OMNIS VCS repository coexisting on the same server, each accessed by their respective versions of the OMNIS VCS.

Opening the VCS

You access the OMNIS VCS from the Tools menu on the main menu bar.

To open the OMNIS VCS

- Select the Tools>>Version Control System menu option on the main menu bar



The VCS Browser displays all your open projects and works in a very similar way to the IDE and SQL Browsers. The VCS menu lets you open and close projects, and perform user administration functions. When you open a project, the VCS Component window opens

showing the contents of that particular project. Note that the VCS shares some of its functionality with the SQL Browser, so you will find the look and feel of the VCS more consistent if you select the same option settings, such as Single Window Mode or Details view, for both.

Setting up a Project

To use the VCS you need to connect to a database via a session, set up the Supervisor user, and create a VCS repository. You can open a session using the SQL Browser or from within the VCS. You can create a repository on a server database, such as Oracle, Sybase, or Informix, or in an OMNIS data file. You can access a repository using ODBC, but not EDA/SQL at present. Connecting to your server or OMNIS database is covered earlier in this manual.

Once you have connected to your database, the VCS operates in the same way, regardless of where the repository is held. However, if you are using Sybase for your VCS repository, you must make sure the transaction log has enough capacity to handle your transactions. See the Sybase documentation for details on the transaction log. You can use the command

```
dbcc checktable (syslogs)
```

while you are running the VCS against your repository to check the status of the transaction log. You must also set the option “select into/bulkcopy” to false with the following remote procedure call

```
sp_dboption <db>, “select into/bulkcopy”, false
```

where <db> is your Sybase database name. The VCS issues an error message and aborts logon if the target database has the “select into/bulkcopy” option set.

Creating a session

To log on to the VCS you either use an existing session or create a new one.

To create a session

- Select VCS>>Modify Session from the VCS Browser menu bar to display the Session Browser
- Select New from the Sessions menu

or, if you have a session that is a suitable template for a new one

- Click on an existing session and select Duplicate from the Sessions menu

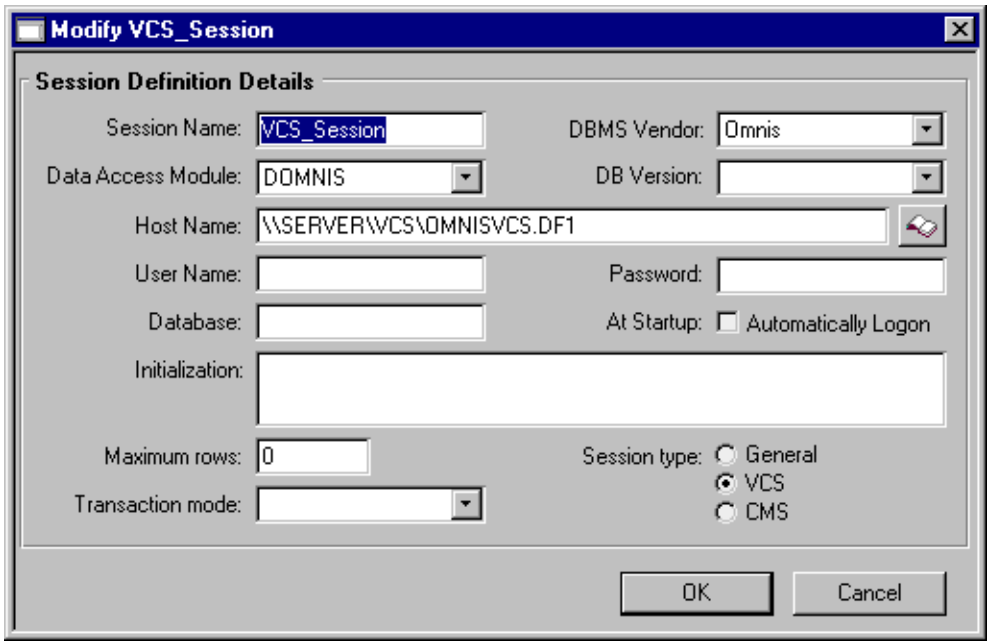
In either case, a new session appears in the Session Browser. You can now click on your session and modify it.

To modify a session

- Select VCS>>Modify Session from the VCS Browser menu bar to display the Session Browser
- Click on a session and select Modify from the Sessions menu

or

- Double-click on a session to modify it



The Modify Session dialog lets you modify the details of the selected session; it is identical to the session definition dialog in the SQL Browser. The information you need to supply depends on the database you want to use; see the *Accessing your Database* chapter for details.

Note that to make a session usable with the VCS, you must select the VCS radio button as the Session type.

- When you have modified the session, click on OK to close the Modify Session dialog

To use an existing session

- Select VCS>>Open from the VCS Browser and select the appropriate session

Signing in to the VCS for the first time

Once you have created and opened a session, next you must log on to your database and sign in to the VCS as the Supervisor user.

The VCS logs on to your database and checks for a VCS repository. If a repository is not found, the VCS asks you whether to install certain VCS resources or tables. If you click on the No button, the log on process is aborted. Otherwise, certain VCS tables are installed so the repository is available for use, and a default user called Supervisor is set up. Once logged on to the repository the Supervisor has access to an option from the VCS menu titled “Remove this Repository”, this option will remove all the tables created by the VCS.

To sign in to the VCS for the first time

When you log on to the OMNIS VCS the Sign in window appears. When you logon for the first time you need to sign in as Supervisor.

- Enter the user name “Supervisor” and the password “password”

Both the user name and password are *case sensitive*, so make sure both words are in the correct case, otherwise you will not be able to logon.

- Click on OK

To ensure a secure system, you should change the Supervisor name and password. You can do this later, but to do it now

- Select VCS>>User Admin from the VCS Browser menu bar

The User Administration window displays one defined user, Supervisor. The fields under the list give details about the selected user, in this case the Supervisor, including the user's name, password, phone extension, department, and status.

- Change the “Supervisor” user name to your name and enter a new password

- Enter any more information you wish to store, and click on the Save Changes button

As a Supervisor, you can allow other users to have Supervisor status so they can create and delete users as well. You will also need to set up the regular users who will check components in and out, and set up preferences. Both of these activities are described later.

- Click on the Finished button to leave the User Administration window

The final step in setting up your project is to check the components in your application in to the VCS. You can check in complete OMNIS libraries to include all the classes in those libraries, you can check in individual classes from any number of different libraries, and you can check in a whole folder hierarchy containing all the necessary files for your application.

Once you have checked all the components in, other users can build a local working version of your library or project using the Build Project option.

Checking in OMNIS Libraries and Classes

All developers working on a project should have access to all the OMNIS classes in your library. Therefore when you create a new project, you should check in all the necessary classes in your library, including any system tables, superclasses, and task classes. The system tables contain information about the fonts, display formats, and so on, used in your library. In particular if the objects in your library use field styles you should remember to check in the #STYLES system table. You can use the Browser Options (press F7/Cmnd-7) to make sure all the classes in your library are displayed in the IDE Browser. In future, if you change the system tables in your local library you must remember to check them back into the VCS along with any other classes you may have changed. You should not check in the #DEBUG system table class that may appear in OMNIS; this exists temporarily for internal use only.

To check an OMNIS library or individual classes into the VCS

- Open the main IDE Browser and your library containing the OMNIS classes you want to check in to the VCS

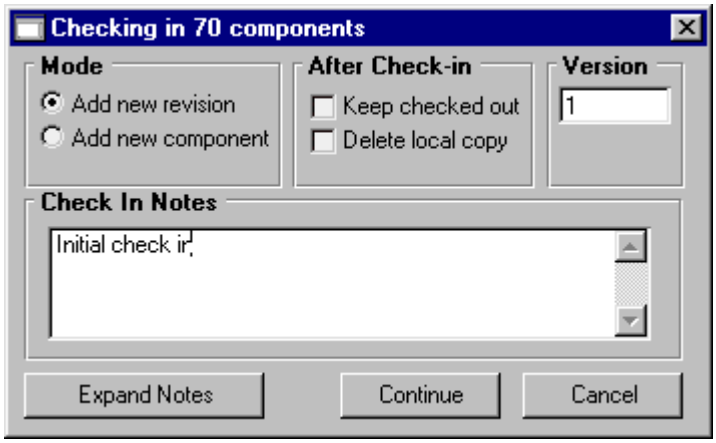
If you want to check in individual OMNIS classes

- Display the classes in your library, either using View>>Down One Level or by double-clicking on your library icon
- Drag and drop your library or selected classes onto the VCS Browser

or you can

- Select your library or classes and click on the VCS button in the IDE Browser toolbar

Whichever method you choose, the Check in components dialog appears. This dialog lets you set the check in options for the selected components. If you check in a library all the classes in the library are checked in automatically.



- Select the Add new component radio button, and type “Initial check in” or something similar in the Check In Notes field

Version Numbers

The version string for each component is currently set to “1”. You can enter a special version in the Version field if you want to use a different numbering scheme, though this is not recommended. All the components in the VCS have both a version and a revision number, each of which is an integer value. The version indicates the major revision or release of a component and typically applies to all the classes in the library, so it’s effectively the version of the library. The revision number indicates a relatively minor change to the component. When you first check in a component, you can assign it a version; the VCS automatically sets this to 1 and the revision number to 0. Each time you check in a component, the VCS assigns a new revision, but the version doesn’t change unless you change it explicitly.

- Click on the Continue button to start checking in

A progress bar shows the number of components added. The VCS queries your database for a project containing the first component. If one does not exist, the VCS asks you to add a new project.

- Click on the Add new project button

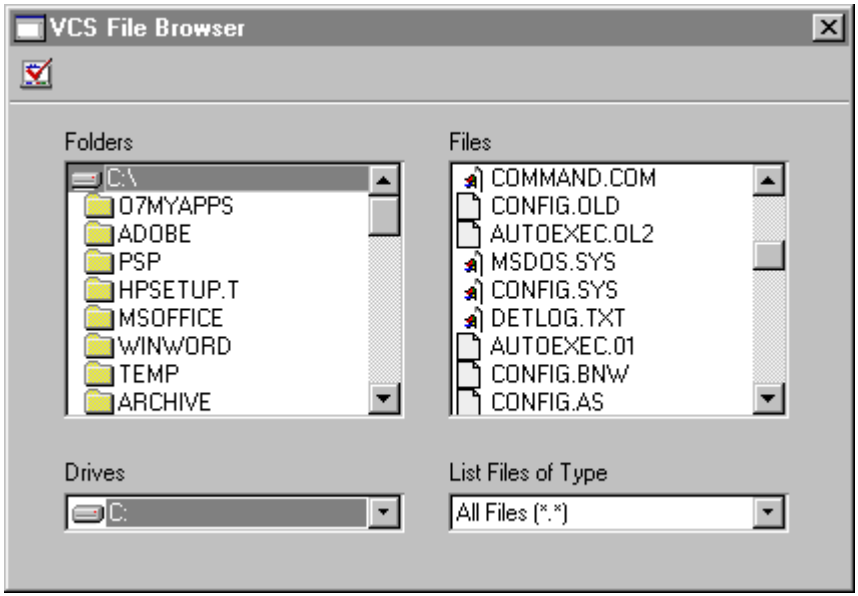
When all the components are checked in, the project appears in the VCS Browser, with the same name as your library. You can rename it if you wish from the Project menu.

Checking in non-OMNIS Components

You can use the OMNIS VCS to control all other types of non-OMNIS components such as external components, text files or Web pages, PDF files, OMNIS data files, and so on. In fact you can use the OMNIS VCS to manage any type of project, including ones that don't contain OMNIS libraries or classes. To check in non-OMNIS components you use the File Browser available within the VCS itself.

To check non-OMNIS components into the VCS

- Open the OMNIS VCS
- Select the VCS>>File Browser option from the VCS menu bar



- Locate the folder containing the files your want to check into the VCS
- Drag a folder or selected files onto the VCS Browser

or you can

- Select the folders or files and click on the Check in button in the File Browser toolbar

If you check in a folder, all the files and *all subfolders and files* within that folder are checked into the VCS. Note that you can check in an OMNIS library file from the VCS File Browser. This is OK if you want to manage a library file as a whole, but if you want access to the classes in the library you should check the library into the VCS from the IDE Browser as described in the previous section.

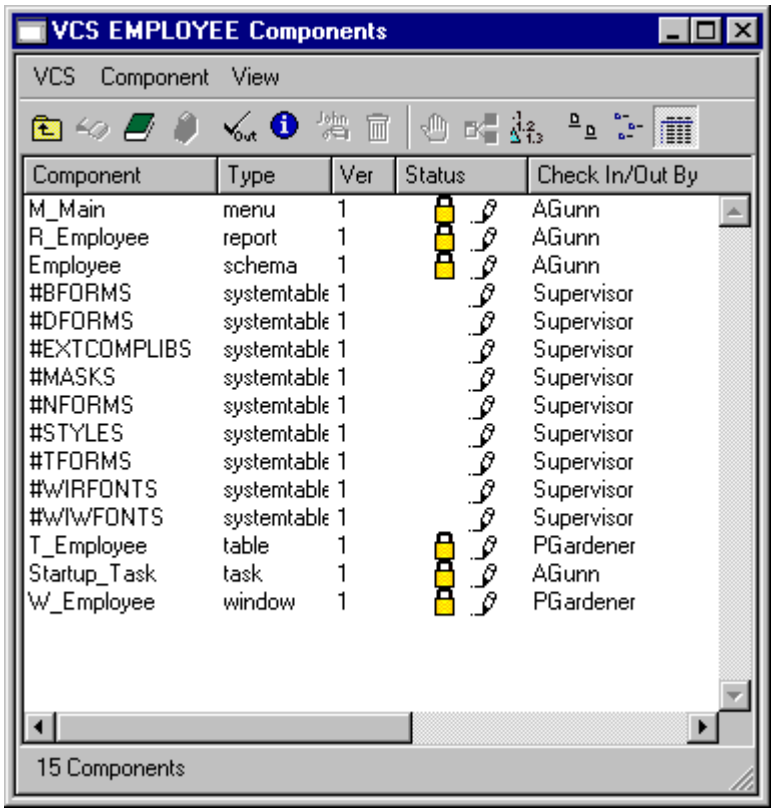
From hereon, the check in process for non-OMNIS components is exactly the same as for OMNIS classes, that is, the Check in components dialog appears which lets you set the check in options for the selected components.

Viewing the Contents of a Project

To see the contents of a project

- Double-click on the project in the VCS Browser

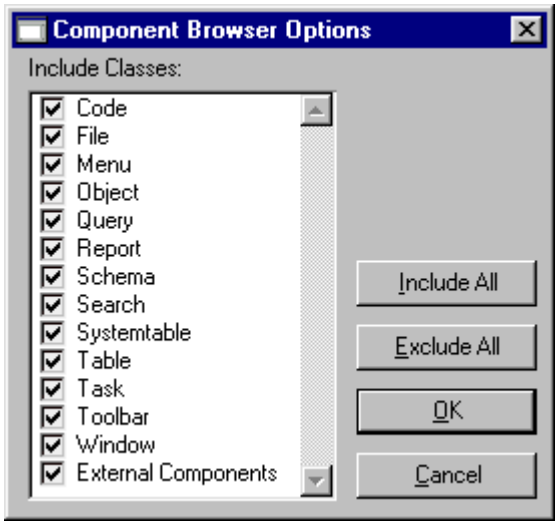
You may find the Details view most useful when viewing the components in a project, since that view shows information such as the version, the status, who has checked each component in or out, and so on. In addition, you can sort the contents of your project by clicking on one of the column headers in the VCS Browser.



The Browser Options lets you specify which components are shown in the VCS Component Browser.

To open the VCS Browser Options

- Select View>>Browser options from the VCS menubar



User Administration

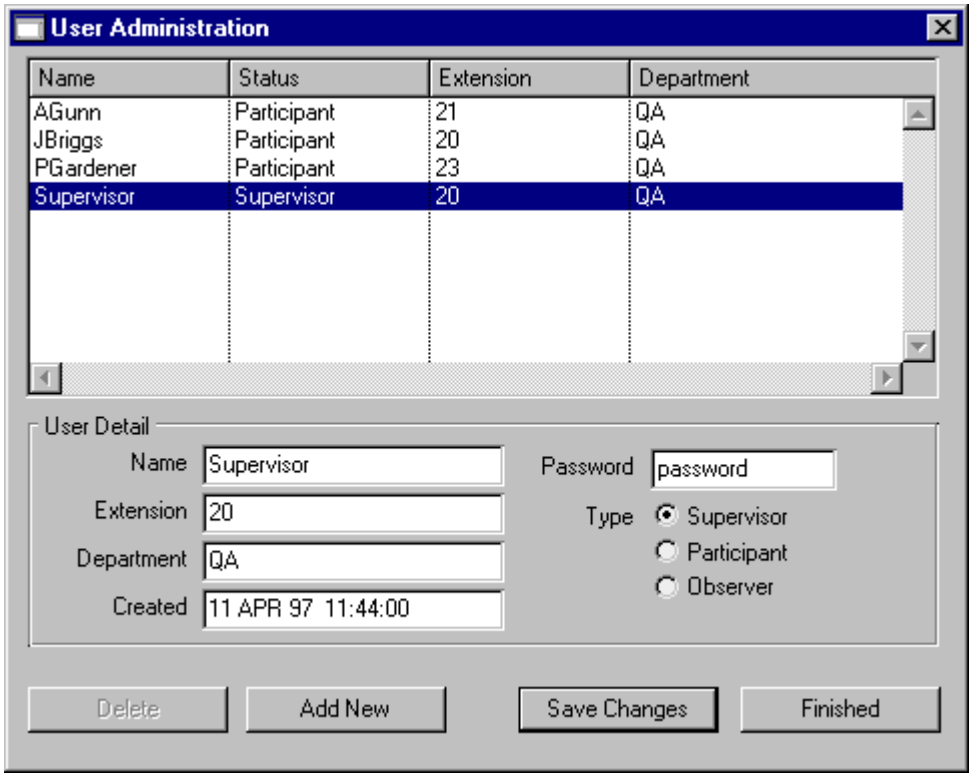
User administration involves managing the users of a project and assigning them privileges for the components in the project. Before developers can start using the project, you need to add each one as a project user and grant the right privileges for the components they need to change.

Adding and Removing Users

Once you sign into the project as Supervisor, you can add, alter and remove users of the project.

To add a new user

- Select VCS>>User Admin



The User Administration window displays a list of existing users, with details about the selected user displayed below it. The Status radio buttons show the three kinds of status a user can have:

– **Observer**
  can see components only

– **Participant**
  check components out and in, but cannot perform user admin

– **Supervisor**
  can do everything, including adding or deleting users and changing user details, and checking components out and in

Only a Supervisor can see a user's password. Only the first Supervisor user can grant Supervisor status to other users.

- Click on the Add New button
- Add the new user information, including the name and password
- Change the type of user as appropriate; a new user is set to Participant by default

- Click on the Save Changes button to save the new user definition

To change an existing user's definition

- Select the user in the list
- Change the user details, and click on the Save changes button

Do not change the status of the Supervisor user, particularly if you have only one user with Supervisor status. If you change the status of the Supervisor user you will no longer be able to manage the project and its users.

To delete a user

- Select the user in the list and click on the Delete button
- Click on the Finished button to leave the User Administration window

Using the VCS

Once the Supervisor has created a project and set up the users, developers or users can start using the VCS. In general, you can

– Sign in to the VCS

– Check out one or more components that you need to work on

– Check those components back into the VCS once you have finished with them

Signing in to the VCS

To sign in to the VCS as a user

- Select Tools>>Version Control System from the main OMNIS menubar to open the VCS Browser
- Select VCS>>Open, or click on the Open VCS button to open the Session Browser
- Click on a session and select Sessions>>Logon to connect to your database
- Enter your user name and password

If you don't have a valid user name and password, you can sign in as a temporary user by checking the Observer check box. An observer can view information about the components stored in a project, but cannot check components in or out, or perform any other VCS tasks. Alternatively, you can check the Create user check box to create a new user and password,

which signs you in with Observer status. A Supervisor then needs to change your user status to Participant or Supervisor if you want to perform any VCS tasks.

- Click on OK to sign in to the VCS

Checking or Copying out Components

When you *check out* a component into a local library, it becomes locked in the VCS, preventing other users from checking it out. A locked component is shown in the VCS with a lock icon. Alternatively, if you *copy out* a component it is not locked in the VCS and other users can check it out if they wish. When copying out a component you can change it in your local library, but you cannot check it back in to the VCS. In practice, copying out is a convenient way of viewing components locally without locking them in the VCS.

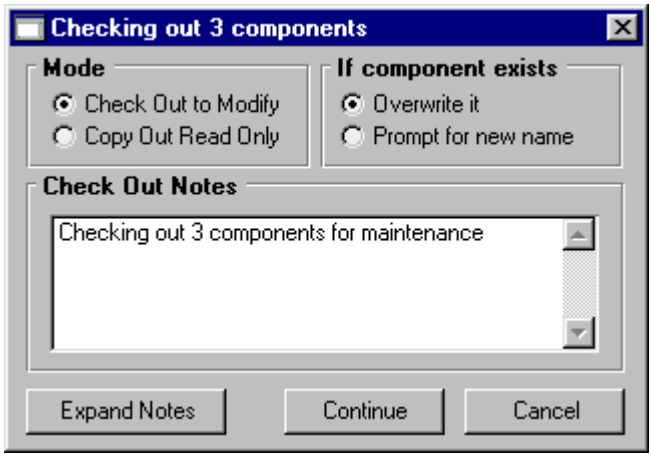
You can check out or copy out multiple components at the same time, but they must all be OMNIS classes or all non-OMNIS components during a single check out process. Components may have the same or different target libraries or folders.

To check out or copy out components

- Display the components in your project by double-clicking on the project in the VCS Browser, or select Down One Level from the context menu
- Select the components in the VCS Components Browser that you want to check out or copy out
- Select the Component>>Check Out menu option, or click on the Check out button in the VCS toolbar

or for OMNIS classes and assuming you can see the IDE Browser

- Drag the classes from the VCS Component window to the appropriate library in the IDE Browser



- In the Check out dialog, select either Check Out or Copy Out and whether the component should overwrite an existing one or prompt for a new name (you can change the default for both these options in the VCS Options)

- If you want, add a description for the checking or copying out process

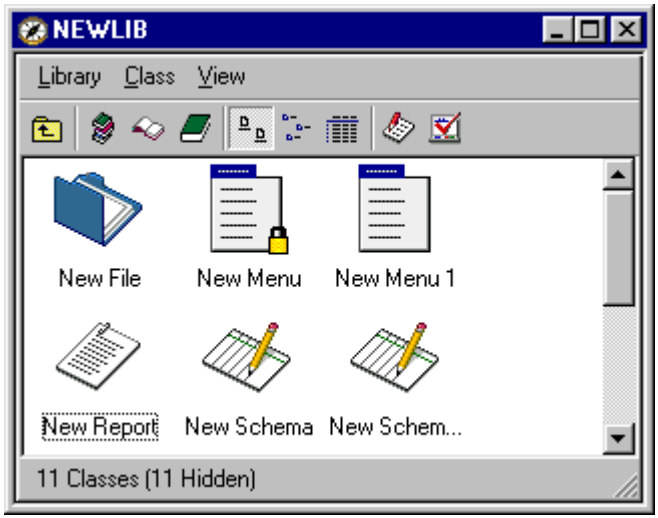
The Expand Notes button can be pressed if individual notes are required for each component in the check out group. Once pressed the window shows an entry for each component.

- Click on Continue

If the VCS cannot locate the original library for an OMNIS class, it will prompt you for a target library. You should select a library, or skip this component. The VCS may prompt you to locate a library for further classes. For non-OMNIS components the VCS prompts you for a destination folder.

If you check out an OMNIS class that has a superclass or belongs to a design task, read-only copies of the superclass(es) and design task are copied out, assuming the Automatically copy out related components preference is enabled.

In the main IDE Browser, OMNIS classes are shown as checked out with a lock icon.



Checking in or Unlocking Components

When you have finished making changes to a component in your local library, you need to check it back into the VCS. This unlocks the component in the current VCS repository and allows other users to make changes or propagate your changes into their local libraries.

Checking in a component releases its lock and makes it available to other users for checking out. The VCS stores the changes in its repository and updates the revision number. You can see the current version and revision by clicking on a component and selecting the Component>>Information menu option.

You can also unlock a component without checking in a new version to make it available for other users. You could do this if you decide you don't want to make any changes to a component after all, or if someone else needs to make immediate changes to a component that you have checked out and is currently locked. In the latter case, you can unlock the component, and check it out again when the other user has made their changes. You can unlock selected components using the Component>>Unlock menu option.

To check in OMNIS classes

- Select the components in your local library in the IDE Browser
- Drag the selected components on to the appropriate project in the VCS Browser

or

- Click on the Check in button on the IDE Browser tool bar

To check in non-OMNIS components

- Open the File Browser from the VCS menu
- Locate the folder containing the files your want to check into the VCS
- Drag a folder or selected files onto the VCS Browser, or select the folders or files and click on the Check in button in the File Browser toolbar

For all types of component, the Check in window appears which lets you set the check in choices for the selected components, as already described in the Setting up a Project section.

You can select one of the following check in modes

– **Add new revision**
adds the component by incrementing the revision number by one, unlocks the component, and updates the check in date, time and other status information

– **Add as new component**
adds the component as a new one, setting the version to 1 and revision to zero, by default

You must set the mode to Add as new component the first time you check a component into the VCS.

The After Check in option lets you decide whether to

– **Keep checked out**
adds the component to the VCS and locks it; in effect, this keeps the component checked out and available for you to make changes

– **Delete local copy**
adds the component to the VCS and deletes the local copy

You can enter a new version in the Version field or accept the default, and regardless of the check in mode, the VCS sets the version to that number.

Building Projects

The OMNIS VCS lets you build a project on your local workstation or any other destination from the components stored in the VCS repository. Building any type of project from the VCS guarantees that all the components in your local copy are up-to-date. In the context of OMNIS application design, building a project means creating a library containing up-to-date classes for you to work on or test; if your application contains multiple libraries, doing a build from the VCS guarantees that your whole application is up-to-date. You can also build a project containing non-OMNIS components and reproduce the original folder hierarchy required in your application. Using the revision labeling features, you can build previous versions of a library or project for comparison, troubleshooting, or debugging.

When you build a project that contains classes from more than one library, the VCS copies all the components to a single library by default. However by setting the Build options you can copy classes to separate libraries, thus maintaining your original library structure.

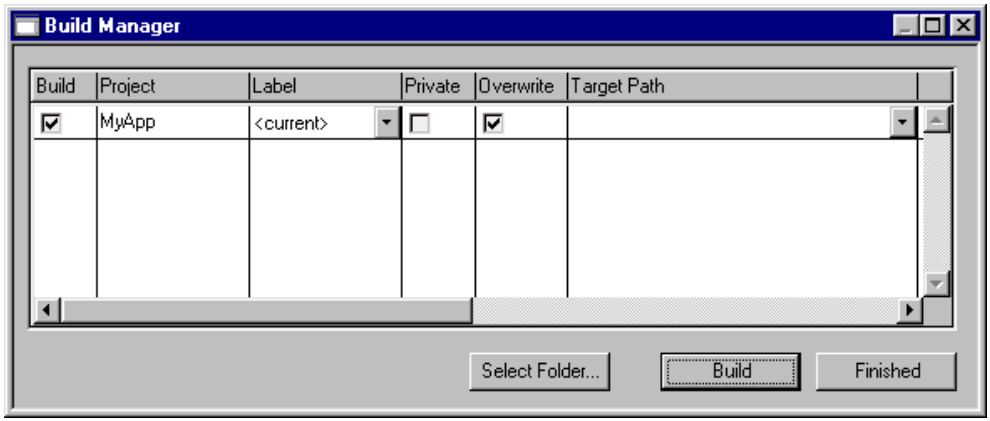
To change the Build preferences

- Select the VCS>>VCS Options menu item in the VCS Browser
- Click on the Build tab and check the Maintain project structure check box

To build a project

Whether or not you are building to one or more libraries

- Select your project in the VCS Browser
- Select Project>>Build Project, or click on the Build project button on the VCS Browser tool bar



The Build Manager lets you configure the project build in detail. It contains one line only if you have chosen to build a project containing OMNIS classes into one library, with the

project name as the library name. If you have set the preference to maintain your library structure, the Build Manager lists each separate library in the project. If your project contains non-OMNIS components you can enter the name and path of the target folder.

- Select the library or project
- Make sure the entry in the Build column for the library is set to True, otherwise the contents of the project will not be included in the build

You can set the following options for a library

– **Label**
selects the version of the library that you want to build; see the Labels section below

– **Private**
if true, makes the library private after it is built; making a library private prevents other users from seeing or changing the contents of a library; for example, the libraries in the OMNIS\Startup folder are private, which means they are not visible in the OMNIS IDE

– **Overwrite Objects**
if true, overwrites all the components in the target library or folder, that is, the VCS removes all the classes in the target library and completely rebuilds the library. Otherwise if false, the VCS updates only the components that are different from the current library or project. This is useful for quickly bringing an existing library up-to-date with the current checked-in classes in the VCS

- Type the path for your library or project in the Target Folder column, or select from any previous build destinations in the droplist. The Select Folder button can be used to select a different folder or create a new folder.

If you are building to multiple libraries, repeat this process for all the libraries in the project that you wish to build. When you have set the options for your libraries or project

- Click on the Build button

The VCS opens the Build Results dialog showing the progress and status of the build.

System Tables

When you build a library, the VCS includes the system tables by default, assuming you have copied them to your project; they contain library-specific settings such as fonts, input masks, field styles, and so on. Remember that when you check in a version of the library, the VCS does not automatically put the system tables into the library; you must specifically show and select them in the IDE Browser just as you do the other classes in your library.

Labels

In long-term projects, the components in your project may undergo many revisions. The VCS tracks these revisions using labels. You can reproduce a particular version of a library using the appropriate label. A label is a string of up to fifty characters that you can assign to a project and each of its components. When a project is complete and you want to release it, you can assign a release name by labeling the project. You can see the label for a project or individual component in its Information window.

To label a project

- Select the project in the VCS Browser
- Select the Project>>Label Project menu option, or click on the Label Project button in the VCS Browser tool bar
- Type in the label and click OK

Once a project has been labeled, the last label assigned is shown in the VCS Project Browser if the browser is in Details view.

When you build a project, you can use labels to select either the latest version of all the components, or an older version. You can also check out a specific revision of a component using its label. See the description of the Check Out process in the section on *Managing Components* below.

If you delete a component that has a label, it remains in the VCS. If you build a project using that label, the VCS finds and includes the component in the build even though you have deleted it from later revisions of the project. The VCS does not include the component in any builds or labels that occur *after* you delete the component. For example, if you label a project and its components on Monday, delete a component from it on Tuesday, and do a build on Wednesday using Monday's label your library *will* include the component. However if you do a build on Wednesday using the most recent project label, your library *will not* include the component you deleted on Tuesday.

Sharing Components between Projects

You may decide to use a single component in many projects, for example, a custom logon window, a client file, or a generic start-and-end-date search component. The VCS can *associate* a component with libraries in different projects, storing the component only once, but building it into different libraries when required. You need only revise the component in one place, rebuild the projects, and the VCS propagates the changes to all the libraries that contain the component.

You must have Supervisor access or be the owner of a component to associate it with a project or library.

To associate a component with a project

- Open the project containing the component, and select the component in the VCS Component window
- Select Component>>Associate, or click on the Associate component button in the VCS Component window tool bar

The Associate option displays a tree list showing all your open projects.

- Select the project with which you want to associate the component, or you can open a project in the list and select a particular library
- Click on the Associate button

If the Maintain project structure Build option is not checked, the component is associated with the single library together with all the other components. Otherwise if the Build option is enabled, the component is associated with a library of the same name as the one containing it. If such a library does not exist, it is created automatically.

To delete an associated component in a specific project

- Open the project containing the associated component
- Click on the component, and select Component>>Delete to break its association with the original component

If the component has a label, it becomes disassociated and a small trash can icon is shown. If you build a project using a label, the component is still available for the build, even though the library or project in question may no longer associate with the component.

Managing Components

The VCS has an extensive array of component management functions, including

– Granting user privileges for components

– Managing revisions of components

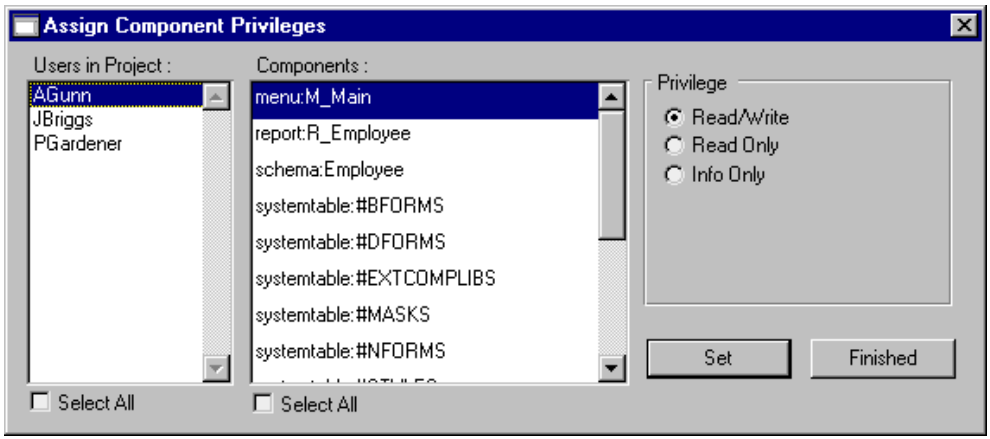
– Deleting and renaming components

Granting User Privileges for Components

If you are the first user to put a component under version control, you are the *owner* of that component. To allow other developers access to a component, you must grant them privileges to it. Without the right privileges, other developers can only view and get information about a component as Observers. Supervisors have all privileges at all times.

To grant component privileges

- Display the components in your project by double-clicking on the project in the VCS Browser, or select Down One Level from the context menu
- Select the set of components for which you want to grant the privileges in the VCS Component window
- Select Component>>Privileges



The Assign Component Privileges window lets you assign privileges for specific components to particular users. You can choose one of the following levels of privilege

– **Read/Write**
the user may check in, check out, copy out, get information on and build libraries with components

– **Read Only**
the user may get information on, build libraries with, and copy out a component into a local library, but cannot check out the component nor check it back in with changes

– **Info Only**
the user may get information only about a component

- Select one or more users and one or more components
- Select one of the levels of privilege and click the Set button to grant the privileges

Assigning the Info Only privilege also revokes existing privileges from a user. Info Only is the default privilege that any user has for a component. The owner of the component must grant the user, including Supervisors, the Read Only or Read/Write privileges necessary for building libraries or changing the component.

By default, the owner of a component starts with Read/Write privileges for that component, and you must have Read/Write privileges to grant privileges for it to other users.

If you select components for which you don't have Read/Write privileges, the VCS disables the radio buttons and Set button. If you select components for some of which you have Read/Write privileges, you can use the radio buttons. The VCS does not however, grant the privileges to the user(s) on those components for which you do not have Read/Write privilege.

Revisions

When you first check in a component, the VCS creates the component in its repository and sets the version to 1 and the revision to zero. Each time you check in a new revision of the component, the VCS stores the new component and increments its revision number. The sequence of changes to a component is stored in its *revision history*.

To see the revision history for a component

- Click on the component in the VCS Component window
- Select the Component>>Revision History menu option, or click on the Revision history button

The Revision History window tells you

– the version string and revision number

– the user who modified the component

– the component name

– and change notes, if any

The revisions for a component are listed in chronological order with the most recent revision at the top. To delete old revisions of a component you can select a revision and click on the Delete Selected and Older button to delete the selected revision and all older revisions below it in the history list.

If you have Read/Write or Read Only privileges for the component, you can copy out the latest revision of the component from the Revision History window by clicking on the Copy Out Revision button. A list of currently open libraries appears, so you can choose which one to copy the component into. Select a library and click on Select.

If a component already exists in the library with the same name as the copy, the VCS prompts for a new name. Keeping the old name overwrites the existing component; entering a new one renames the copy.

Component Services

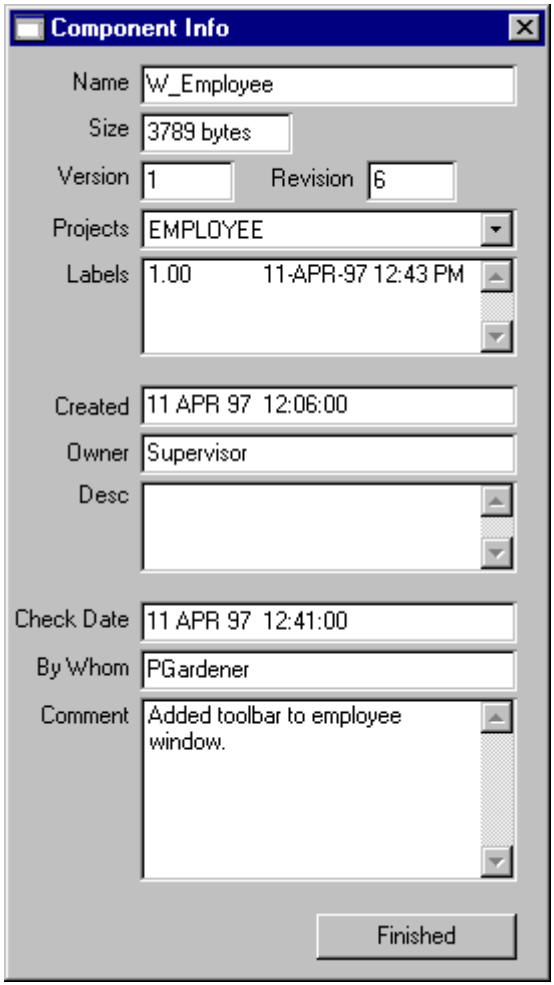
The Component menu in the VCS Component window gives you information about a component, and lets you rename, delete, or unlock a component. If you select more than one component, the VCS acts on the first component only.

To get Info about a component, or rename, delete, or unlock one

- Select the component in the VCS Component window
- Select the appropriate menu option from the Component menu, or click on the Info, Rename, or Delete button in the VCS Component window tool bar

Get Information

This window displays the information about the component, including the labels, the revision state, and the check out history.



Deleting Components

To delete a component, you must own the component and it must be checked in. The Delete option checks whether the component belongs to more than one library and whether any labels apply to it. If not, the VCS prompts for confirmation that you want to delete the component. If it does belong to more than one library, the VCS displays a list of associated libraries, letting you choose the ones from which to remove the component.

Renaming Components

To rename a component, you must own the component and it must be checked in. You can rename any component in the VCS, but renaming OMNIS classes requires some additional care because of the potential dependencies of some classes on the class you want to rename. That is, before you rename a class in the VCS you must change all references to it in any other classes that refer to the class. You do this by checking out all the relevant classes to the IDE Browser and using the OMNIS Find and Replace tool.

To rename a class

- Check out the class and any other classes that contain methods which refer to the component
- Change the name of the component using the IDE Browser, and answer Yes when you are prompted for whether or not you want to use Find and Replace; this renames the component and changes all references to it in any other components
- Check in all the components *except* for the renamed one
- Click on the component in the VCS Component window, and select the Component>>Unlock menu option to release its lock
- Click on the component again when the VCS Component window has refreshed
- Select the Component>>Rename menu option, or click on the Rename component button
- Enter a new name in the Rename Component dialog, and click OK

Unlocking Components

You can unlock a locked component by clicking on it and selecting the Component>>Unlock menu option. A warning message advises you that having done this, you will not be able to check in a previously checked out version of the component.

Setting VCS Options

You can set your individual preferences for the VCS using the VCS>>VCS Options menu option. You can set Display, Check out, Check in, and Build options in the VCS Preferences window.

To set your VCS options

- Select the VCS>>VCS Options menu item in the VCS Browser

Display Options

The Refresh after Check In/Out option lets you control whether or not the VCS redraws its window automatically after checking components in or out. Otherwise, you can refresh the window manually using the VCS>>Refresh menu option.

Check Out Options

The first option, Default mode is Check Out Modifiable copies, determines whether or not a component is checked out or copied out by default. The second option, Default replacements to overwrite existing items, determines whether or not an existing component in your target library is overwritten when a component is checked out. If this option is not set, you are prompted to overwrite or rename the component.

The third option, Automatically copy out related components, relates to OMNIS classes that have superclasses and design tasks. If you have a class that is a subclass of another class, and you check out the subclass, without this option the superclass(es) will not be copied out; in this case, you will not see any fields and methods that are contained in the superclass. If this option is set, read-only copies of the superclass(es) are copied out, as well as the design task for the classes.

Check In Options

The first option, If Target Component Does Not Exist, controls what happens when you check in a new component; by default components are added to a project automatically, otherwise you can force the VCS to prompt you for instructions. The second option, When Component Version Number Changes, controls revision numbers for new versions of a component; by default revision numbers are set to zero for new versions, otherwise you can force the VCS to continue to increment revision numbers regardless of the component version. Version and revision numbers are described earlier in this chapter.

Build Options

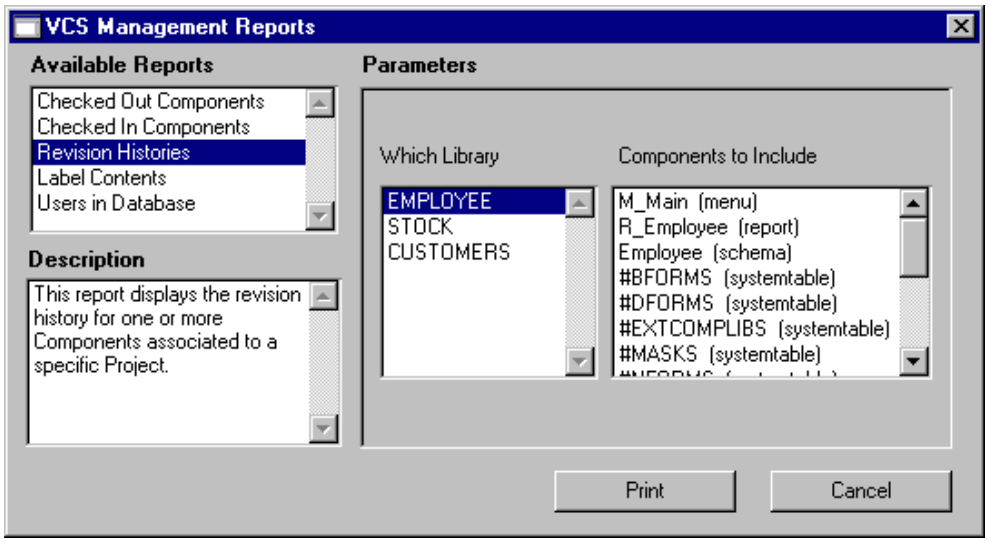
The Maintain Project Structure option controls whether or not a build retains the library structure of the components within a project. If this option is checked, components are built into separate libraries mirroring the original library structure, otherwise all the components in a project are built into a single library.

Reports

The VCS has a number of reports available to those with Supervisor status. You can access them using the VCS>>Reports menu option. Users without supervisor status do not have access to VCS reports and will find this menu option is grayed out.

To use the VCS reports

- Select the VCS>>Reports menu option in the VCS Browser



The VCS Management Reports dialog lets you select a report. The Parameters pane and Description field will change depending on the report you click on.

- Select a report and set up its parameters
- Click on the Print button to generate the report
- Select the required destination, and click on OK

Chapter 15—Deploying your Application

Once you have developed an application, you will need to deploy it to your end-users. OMNIS Software provides a runtime version of the OMNIS Studio executable or program, which contains all the necessary elements to run an application, but does not include the developer-only parts, such as the OMNIS IDE tools. The runtime version for each platform is available on the OMNIS Studio CD.

You may find it convenient to deploy your applications, including the OMNIS runtime for the required platform, using one of the standard Installer products such as InstallVise or InstallShield.

Before an end-user can run the application, it must be serialized. It is your responsibility to obtain the required runtime serial numbers from OMNIS Software for distribution with your application.

Serialization

OMNIS Studio is serialized the first time you run it, rather than during installation. When you startup OMNIS a serialization dialog appears, prompting you for the your name, company, and serial number. You can update or re-serialize OMNIS at any time using the Change Serial Number option in the Tools menu on the main menubar.

Changing the Serialization Dialog

When you deploy your application your end-users need to serialize the OMNIS runtime at startup, therefore you may want to change the standard OMNIS serialization dialog to suit your needs. The $serialize() method opens the standard serialization dialog, and also lets you modify its appearance.

– **$serialize**([bGenericLogo,cTitle,iBitmapID])
opens the OMNIS serialization dialog; you can pass your own title and bitmap, otherwise if bGenericLogo is kFalse (the default) the OMNIS orb is displayed, or if kTrue a generic Serialize logo is displayed; cTitle replaces the default title

```
Do $root.$prefs.$serialize()
; displays the standard dialog
Do $root.$prefs.$serialize(kTrue)
; displays the same dialog but with a key bitmap replacing the OMNIS
   one
```

Index

OMNIS

Using OMNIS Studio

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.