

Developing Web Applications in Omnis

Omnis Software

October 2000

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

© Omnis Software 2000. All rights reserved.
Portions © Copyright Microsoft Corporation.

Omnis® is a registered trademark, and Omnis Studio and Omnis Web Client are trademarks of Omnis Software.

Microsoft, Windows, Windows 95, Windows NT, Win32, Internet Explorer, and FrontPage are either trademarks or registered trademarks of Microsoft Corporation in the US and other countries.

Netscape, and Netscape Navigator are registered trademarks of Netscape Communications Corporation in the US and other countries.

Apple, the Apple logo, AppleTalk, and Macintosh are registered trademarks and MacOS, Power Macintosh and PowerPC are trademarks of Apple Computer, Inc.

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

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

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

HP-UX is a trademark of Hewlett Packard.

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

Acrobat is a trademark of Adobe Systems, Inc.

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

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

INFORMIX is a registered trademark of Informix Software, Inc.

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

CodeWarrior is a trade mark of Metrowerks, Inc.

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

Table of Contents

ABOUT THIS MANUAL.....	8
INTRODUCTION.....	8
<i>Create all types of web applications.....</i>	<i>8</i>
<i>Scalable Multi-Tier Environment</i>	<i>9</i>
<i>Unique Web Client and Components.....</i>	<i>9</i>
<i>Optimized Application Server and Data Access.....</i>	<i>9</i>
CHAPTER 1–OMNIS WEB CLIENT	11
APPLICATION ARCHITECTURE.....	12
THE CLIENT.....	13
<i>Web Browser.....</i>	<i>13</i>
<i>Omnis Web Client.....</i>	<i>13</i>
<i>Omnis Web Wizards and Components.....</i>	<i>14</i>
<i>Installing the Web Client for Development.....</i>	<i>15</i>
THE SERVER.....	15
<i>The Web Server.....</i>	<i>15</i>
<i>The Web Server Plug-in.....</i>	<i>15</i>
<i>The Omnis Server</i>	<i>16</i>
<i>Omnis Application.....</i>	<i>16</i>
<i>Database Server</i>	<i>16</i>
STANDARD HTML FORMS	17
CHAPTER 2–OMNIS WEB CLIENT TUTORIAL.....	18
DESIGNING THE REMOTE FORM.....	19
<i>Using the Sidebar Remote Form Wizard</i>	<i>20</i>
<i>Editing Form Fields.....</i>	<i>33</i>
<i>Adding Variables and Methods to the Form.....</i>	<i>34</i>
<i>Events and Adding Methods to Form Fields</i>	<i>40</i>
<i>Adding a Picture Field</i>	<i>41</i>
<i>Testing the Form and Browsing the Books.....</i>	<i>43</i>
<i>Debugging the Form.....</i>	<i>43</i>
DEPLOYING THE REMOTE FORM.....	44
<i>Omnis Server</i>	<i>44</i>
<i>Web Server and HTML files.....</i>	<i>46</i>
VIEWING THE OMNIS WEB CLIENT EXAMPLE LIBRARY	48

CHAPTER 3—REMOTE FORMS.....49

CREATING REMOTE FORMS USING WIZARDS	49
<i>Plain Form Wizard</i>	51
<i>Tabs Form Wizard</i>	52
<i>Wizard Form Wizard</i>	57
<i>Submit Form Wizard</i>	58
<i>Password Form Wizard</i>	60
<i>Sidebar Form Wizard</i>	62
<i>Sidebar with pages Form Wizard</i>	65
<i>Multiform Form Wizard</i>	66
<i>New Remote Form Template</i>	69
DEBUGGING AND TESTING REMOTE FORMS	69
ICONS AND ICON PAGES	69
<i>Creating your own Icons and Icon Pages</i>	70
CURSORS	70
EVENTS	70
<i>Standard Field Events</i>	71
<i>Pushbuttons and Lists</i>	71
<i>Remote Form Events</i>	72
PROGRAMMING REMOTE FORMS	72
<i>Optimizing Data Handling</i>	73
<i>Open windows and User Prompts</i>	73
<i>Redrawing Remote forms</i>	74
<i>Opening a Browser window on the Client</i>	74
FORM CACHING	75
<i>Server Form Caching</i>	75
<i>Client Form Caching</i>	75
CLIENT METHOD EXECUTION	76
MULTIPLE FORMS	77

CHAPTER 4—REMOTE FORM COMPONENTS78

<i>Web Component Properties</i>	79
BORDER	80
CALENDAR	80
CHECK BOX	82
CLOCK	83
COMBO BOX	85
DROPLIST	85
EDIT FIELDS	86
FADE	87
FILE READ/WRITE	89
<i>Reading files</i>	90
<i>Writing files</i>	90
<i>Events</i>	91

GIF	92
HEADED LIST	93
HOTPIC	97
ICON ARRAY	99
INFO	100
JPEG	105
LABEL	106
LIST	107
MARQUEE.....	108
PAGE PANE	108
PICTURE	109
PORT	109
<i>Getting a list of Port Names</i>	111
<i>Creating a Port</i>	111
<i>Opening a Port</i>	112
<i>Reading from a port</i>	112
<i>Writing to a Port</i>	114
<i>Closing the Port</i>	114
<i>Multi Port Support</i>	115
<i>Aborting Read/Write Operations</i>	115
<i>Error Handling</i>	116
PRINT	116
PROGRESS	118
PUSHBUTTON AND BUTTON AREA.....	119
QUICKTIME.....	120
RADIO GROUP	121
ROLL BUTTON.....	122
SIDEBAR	122
SLIDER.....	123
TAB BAR.....	125
TILE.....	127
TIMER.....	127
TRANSBUTTON	128
TREE LIST	129
WASH.....	130
CHAPTER 5—REMOTE TASKS.....	131
CREATING REMOTE TASK CLASSES USING WIZARDS.....	131
<i>Plain Remote Task Wizard</i>	132
<i>Monitor Remote Task Wizard</i>	133
<i>HTML Report Task Wizard</i>	134
<i>Submit Remote Task Wizard</i>	135
CREATING A NEW REMOTE TASK CLASS	136
REMOTE TASK INSTANCES	136

EVENTS	137
CLIENT ACCESS PROPERTIES	137
<i>Timeouts</i>	138
<i>Client Connections</i>	138
SECURE SOCKETS	139
CHAPTER 6—USING STANDARD HTML FORMS....	140
<i>Task Wizards and \$construct</i>	142
CHAPTER 7—MULTI-THREADED SERVER	144
MULTIPLE METHOD STACKS	144
USING THE SERVER	145
DATABASE ACCESS	146
<i>Omnis datafile access</i>	146
OMNIS SERVER COMMANDS.....	146
<i>Start server</i>	146
<i>Stop server</i>	147
<i>Begin and End critical block</i>	147
<i>Yield to other threads</i>	148
<i>Commands which are not available to a client</i>	148
CHAPTER 8—SERVER LOAD SHARING.....	149
ENABLING LOAD SHARING	150
<i>On the LSP servers</i>	150
<i>Load Sharing Mechanism</i>	150
USING VERSION 2.X WEB CLIENTS	151
CHAPTER 9—DEPLOYING YOUR OMNIS WEB APPLICATION	152
EDITING YOUR HTML PAGES.....	153
<i>Embedding the ActiveX Web Client</i>	153
<i>Embedding the Netscape Plug-in Web Client</i>	155
<i>Autodetecting Browser/Platform</i>	156
<i>CGI</i>	157
OMNIS SERVER CONFIGURATION.....	157
<i>Setting the Omnis Port Number</i>	157
<i>NT Server Setup</i>	158
WEB SERVER CONFIGURATION.....	159
<i>Installing the Web Server Plug-in</i>	159
<i>ISP Web Hosting</i>	160
<i>Secure Sockets</i>	160
REMOTE FORM COMPONENTS AND INSTALLERS.....	161
<i>Web Client Components</i>	161
<i>Creating your own Web Client Installers</i>	162
<i>Auto downloading the Web Client installer</i>	163

AUTOMATIC INSTALLATION.....166

Configuring your Omnis Web Server to support Automatic Installers 166

CHAPTER 10–TROUBLESHOOTING GUIDE AND FAQ 167

GENERAL.....167

REMOTE FORMS167

BROWSER168

OMNIS SERVER.....169

INDEX170

About This Manual

This manual describes how you use Omnis Studio™ and the Omnis Web Client™ to develop applications and business solutions for the internet or your intranet. This manual:

- ❑ Provides an overview of the Omnis web client.
- ❑ Describes how to install and configure the Omnis web client and the Omnis Server.
- ❑ Describes the Remote form and Task wizards.
- ❑ Describes how to create and modify remote forms and tasks.
- ❑ Provides a description of all the Omnis web components including edit fields, lists, tab bars, sidebars, picture fields, calendar and clock controls, and so on.
- ❑ Describes how you can create Html forms for direct access to your database.
- ❑ Describes how you can optimize the Omnis Server for multi-threading and load sharing.

Introduction

Create all types of web applications

Using Omnis Studio and the Omnis web client you can create many types of application that run on the internet or your company's intranet. In addition, you can take an existing Omnis application, even one developed in a previous version of Omnis, and add web functionality to broaden the scope and appeal of your Omnis solution. The Omnis web client lets you embed your application into a standard web page, and allows anyone with a web browser, to access your data or application from anywhere in the world and whenever they want.

As an application developer you have many choices about how you can deliver your Omnis solutions, but using the Omnis web client opens up many new areas of development in business-to-business and consumer markets. With the emergence of the Application Service Provider (ASP) model, it is possible to use Omnis to create all types of business applications, including e-commerce, supply chain solutions, personal and customer relationship management, company resource planning, and so on, and provide them via the web on a monthly or yearly subscription basis. In the broader context of the business-to-consumer market, you can use the Omnis web client to create websites to deliver all types of service including online banking and shopping, travel and leisure information, picture and photo libraries, and many other services. The Omnis web client delivers highly sophisticated web-based solutions and executes at high speed over normal 56k or ISDN

lines, and, in the very near future, broadband ‘always-on’ internet connections will further improve performance.

Scalable Multi-Tier Environment

Omnis Studio is one of the most powerful, scalable, and flexible Rapid Application Development (RAD) tools available today. In the context of web application development, this means you can develop all types of application and deploy them to all levels of your business or organisation, across many locations, for employees or customers alike. Furthermore, you can extend your existing two-tier or client-server applications using Omnis Studio to take advantage of distributed computing and multi-tier architectures, providing access to your data and services across the internet or your intranet via web browsers and mobile clients. Your development and deployment options are further enhanced because Omnis Studio is entirely cross-platform, which means you can develop your web application on your choice of OS (Windows, Linux, or Mac) and deploy it using a range of different client operating systems, application servers, and remote database options.

Unique Web Client and Components

Omnis Studio provides thin client and ultra-thin client access to your web application via the patent-pending Omnis web client and standard Html forms. The Omnis web client is a web browser plug-in, either an ActiveX or Netscape plug-in, that lets you embed visually rich and interactive forms into standard Html pages placed on a web server. The web client allows methods to be executed on the client machine which optimizes and enhances the user interface.

The web client itself is around 500k so is easily downloaded from the internet or across your local network. Omnis Studio provides over 30 web components that you can include in your web forms, including standard entry fields, lists, pushbuttons, picture fields, tree lists, sidebars, tab bars, multi-media players, and so on. The standard web components are installed along with the web client, while others are downloaded automatically when they are needed in a particular form. The component interface is open so you can create your own web components and include them in your forms. Furthermore, the auto-download process ensures that the web client engine and all web components are kept entirely up-to-date on the client, since the latest versions are downloaded automatically from the Omnis Server.

Optimized Application Server and Data Access

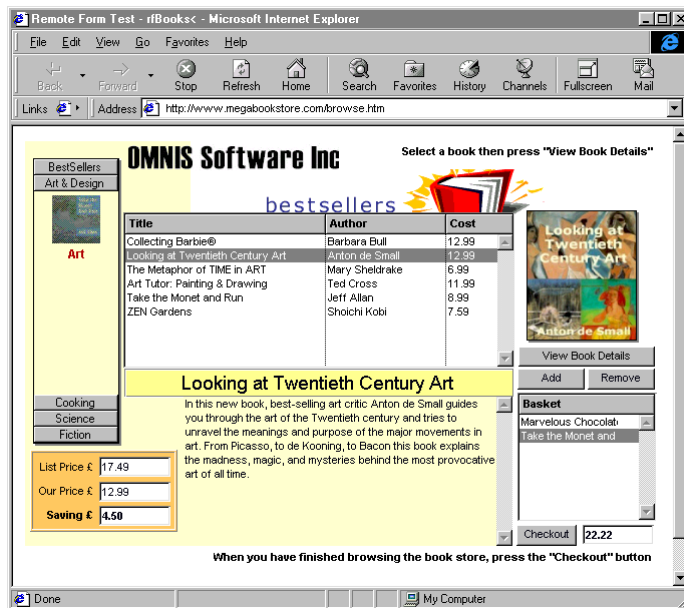
The server side of your Omnis web application comprises a standard web server, the application server containing the Omnis runtime engine and your Omnis library, and your database server. All these parts need not run on the same machine, but typically would be on the same LAN or subnet, communicating via TCP/IP. The web server and the Omnis Server can be located on a Win32 or Linux machine.

The web server would store your entire web, including any Html pages containing the Omnis web client and your remote forms. The Omnis library would contain all the form class definitions, business rules, and application logic, while the database server could be either an Omnis database (data file) or an industry-standard remote database server, such as Oracle, Sybase, DB2, Informix, or any ODBC compliant database. The Omnis Server or runtime engine is a fully multi-threaded server that runs your Omnis library, executes all the business logic, and handles all client requests to-and-from the web server and remote database. The Omnis server is serialized with a serial number that allows a specific number of concurrent users, that is, the number of different users is unlimited but only a specific number of users are allowed to access your application at one time.

Chapter 1–Omnis Web Client

Using Omnis Studio and the Omnis web client you can create any type of application and deploy it to the internet or your intranet. Omnis Studio lets you design remote forms that users can access and display in a web browser. A *remote form* is an Omnis class that contains all the necessary components to display your data and allows users to interact with your application and database via the web. You can adapt an existing Omnis Studio application for web access by adding remote forms to your Omnis application, and adding one or two components to your existing web server and network infra-structure.

The following screenshot shows a web browser containing an example application for browsing books in an on-line bookstore. The form, displayed in the browser using the Omnis web client, contains many types of components including Sidebar, various list components, form fields, buttons, check boxes, and so on.



Note that this Bookstore example library and several other web client example libraries are available in the Welcome Application.

Application Architecture

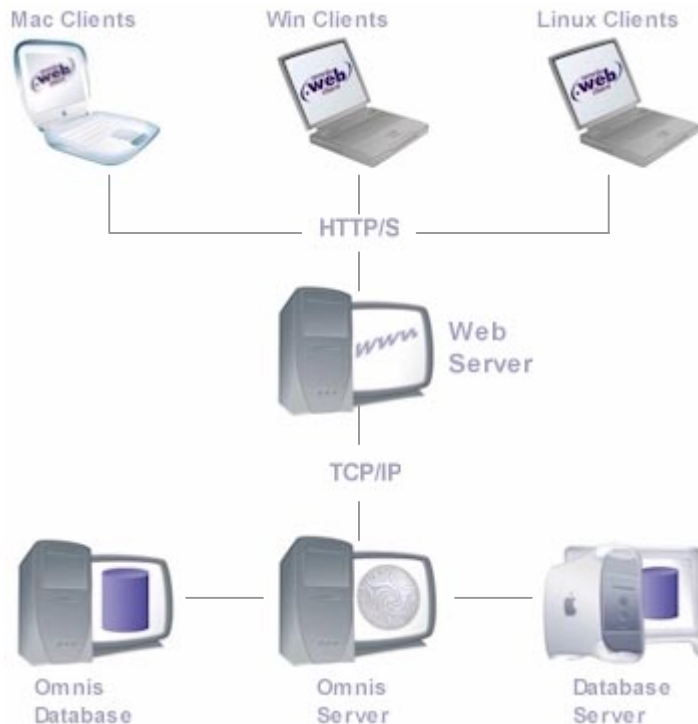
An Omnis web application has two main layers or parts, *the Client* or presentation layer and *the Server* or application and data access layer, although in practice these layers may contain several other parts to complete the whole multi-tiered environment.

❑ The Client

The client can be a Windows, Mac, or Linux desktop machine running an Internet Explorer or Netscape web browser. The user accesses your Html page(s) containing the Omnis web client, which is either an ActiveX or Netscape plug-in. The Omnis web client displays the remote form(s) stored in your Omnis library located on the Omnis Server.

❑ The Server

The server side of your Omnis web application comprises a standard web server, the Omnis Server, and your remote database server or Omnis database. Your web server CGI folder contains an Omnis extension that handles the communication between the client(s) and the Omnis Server. The Omnis Server contains the Omnis runtime engine and your Omnis library containing the remote form classes and business logic.



The Client

To access your Omnis web application, the client requires a standard web browser. The first time a user accesses your application they need to download and install the Omnis web client, which you can provide on your website, or under Windows only, you can allow users to download and install the web client using an automated installer. Having installed the web client, the user would navigate to the appropriate page(s) on your website containing your Omnis web application. This section summarizes the components required on the client machine.

Web Browser

The Omnis web client is available as either an ActiveX or Netscape plug-in, so the client's web browser must support one or other of these plug-ins. For example, users can access a page containing the ActiveX web client using Windows Internet Explorer™ under Windows, or the Netscape plug-in using Netscape Navigator™. When you deploy your application to the web you need to take account of the different browsers in use and adapt your html pages containing the Omnis web client accordingly.

Omnis Web Client

The Omnis web client is a browser plug-in that allows 'thin client' access to applications residing on a remote server alongside a standard web browser. The web client plug-in is an ActiveX or Netscape plug-in that is placed in a standard html page. You can use the Omnis web client in any development environment that supports ActiveX, including Omnis itself and many other development environments. On a Windows client machine, the Omnis web client uses the Wininet API to communicate with the Omnis web server plug-in via HTTP, or HTTPS if secure transmission is required. Communications between the Omnis web server plug-in and the Omnis Runtime and library is via TCP/IP.

Under Windows, note that the Omnis web client could potentially be placed into any environment which supports ActiveX, including applications written in other development tools. The Omnis web client ActiveX could be placed inside a standard Omnis window!

Web Client Installation

To use an Omnis web application containing the Omnis web client, a user must download and install the Omnis web client. This is a one-time operation. You can provide a link on your website to allow users to download the Omnis web client installer. Installers for the Omnis web client are provided on the Omnis Studio CD, and the latest installers are also freely available on the Omnis website. Under Windows only, it is possible to allow users to download and fully install the web client automatically when they first visit a web page containing the web client plug-in; installers for this type of installation are also provided on the Omnis Studio CD.

The standard web client package includes the web client engine and a basic set of web components. When the user displays a remote form containing any additional or out-of-date components, those components are downloaded automatically to the client machine. Similarly, the web client engine is also updated automatically on the client machine. Alternatively, you can make your own custom web client installers containing all the components required for your web application.

Establishing a Connection

When the user accesses your Omnis web application in their browser, the Omnis web client is activated and a connection is established with the server plug-in on your web server (see the Server section below). Once a connection is established, the Omnis web client receives the class data of the remote form plus all the instance variable data, icon pages, font tables, etc., which it gets from your Omnis library. The Omnis web client instantiates the visual instance of the remote form and all of its components, and the user can now use the form.

Events and Method Execution on the Client

You can monitor and control what happens in the remote form by monitoring the *events* triggered by the user on the client. For example, an event may be generated when the user clicks on a form field, or when they select a line in a list, press a button, or tab from one field to another. You can detect and respond to an event by creating an *event handling method*, stored with the object in your remote form. By default, all events are turned off in a remote form, but you can enable specific events for each web component.

You can handle events on the client by executing the event handling method on the client machine, or an event may call and execute a method on the Omnis Server. The method could do anything you want; for example, it could perform a simple calculation, change the properties of an object in the form, or it could send the current data in the form back to the Omnis Server to be saved in a remote server database. When you design your application, you can decide which events are to be handled on the client or the server, and you need to consider what impact the traffic generated by events will have on the user experience and the general performance of your application. Details about handling events and tips for optimizing your application are described later in this manual.

Omnis Web Wizards and Components

Omnis Studio has many different wizards and templates for creating all types of remote form. In addition, the Component Store contains over 30 different web components that let you create very interactive, feature-rich environments for your web applications. These include pushbuttons, single and multi-line edit fields, radio buttons, check boxes, sidebars, list boxes, drop lists, and combo boxes, and many multi-media style components.

Installing the Web Client for Development

You must install the Omnis web client to design and test your remote forms in Omnis Studio. You can test remote forms in design mode using Ctrl/Cmnd-T, but this will only work if you have installed the Omnis web client. When you install Omnis Studio you are asked to install the web client. There is an installer for the Omnis web client on the Omnis Studio CD in the Webclient/Client folder. There is a separate installer for the ActiveX and Netscape plug-in which installs the web client ready for you to start designing and testing your remote forms.

The Server

The server side of your Omnis web application comprises a web server, a web server plug-in supplied by Omnis, the Omnis runtime or executable, your Omnis application, and your data source, either an Omnis database or proprietary server database. All these various server pieces may or may not be located on the same machine, but typically they would be on the same LAN. This section summarizes the components you need on the server side of your Omnis web application.

The Web Server

The Omnis web client requires a stand-alone, industry-standard web server (Win32, MacOSX, or Linux). Web server software is available from many different sources, such as Apache which is included in some Linux installations, and you can download web server software from the Internet. Using your favorite web search engine, you can search for “web server software” to find many hundreds of sources for a suitable web server.

The web server manages the html pages containing the Omnis web client, giving access to your Omnis application. Your web server also requires a web server plug-in, supplied by Omnis, that handles communications between the client and your Omnis application. See the next section.

Note that the Omnis Server, described below, **is not a web server**. The Omnis Server does not serve up html pages, rather your Omnis web application requires a separate web server containing your Html pages, as described above.

The Web Server Plug-in

The web server plug-in or extension is a piece of software supplied by Omnis that sits on your web server, and listens for and queues any requests from the Omnis web client in the user’s browser. Similarly, it passes any results sent from the Omnis Server back to the Omnis web client in the user’s browser.

The web server plug-in is typically installed inside the cgi-bin or scripts folder of your web server. The Omnis web client supports Microsoft ISAPI, CGI, and Apache, and a separate

server plug-in is supplied for each interface. You can find the web server plug-ins in the Omnis Studio installed tree in the Omnis/Webclient/Server/Webserver folder. If your web server does not support the ISAPI interface, you will have to use the `nph-omniscgi.exe` cgi program. In future other interfaces may be supported, such as Netscape Server. The source code for the web server plug-in is available on the Omnis website www.omnis.net so you can create plug-ins for your own platform and interface.

The Omnis Server

To complete your web application, the server-side requires the Omnis Server, that is, an Omnis Runtime executable and your Omnis library containing your remote forms, as well as any necessary business logic and reporting capabilities. The Omnis Runtime and your library can reside on a Win32 or Linux server, local to your web server.

The Omnis runtime must be serialized for a specific number of concurrent users. For example, if your Omnis runtime is serialized with a 100 user serial number any number of people can access your application, but only 100 users can connect to the Omnis Server at a time. The serial number required for the Omnis web client is a special web serial number containing a “W”, a standard multi-user serial number containing an “M” will not work in this case.

Omnis Application

Your remote forms and any necessary business logic for your web application are stored in an Omnis library file. Your remote forms are stored as Omnis *remote form classes*. In addition you need to create a *remote task class* to handle the connection to each client. These classes are described in more detail later in this manual. Designing Omnis libraries is described in detail in the *Using Omnis Studio* manual.

Omnis remote form classes are designed using the Omnis Studio development environment. You can debug or test your remote forms during development using your own web browser. There are many types of fields and components you can use in your remote forms. The web components are implemented as external components, and include pushbuttons, single and multi-line edit fields, radio buttons, check boxes, sidebars, list boxes, drop lists, and combo boxes. In addition, you can use the built-in page pane component in your remote forms. You can create your own components as external components (using C++) and use them in your forms. The remote form class and its components are entirely cross-platform, that is, Win32, Mac, and Linux web browsers can access the same forms stored in the same Omnis library.

Database Server

The Omnis Server can connect to Oracle (7 & 8), DB2, Sybase, and many others server databases via ODBC using the Data Access Modules (DAMs). Please see the *Omnis Programming* manual for full details about accessing your server database using the Omnis DAMs.

Standard HTML Forms

As an alternative to using the Omnis web client and remote forms, Omnis Studio lets you interact with your Omnis application and database over the Internet using standard html forms. In this case, your html forms connect to an Omnis remote task class direct, and no remote forms are required for this type of interaction. See the *Remote Tasks* chapter for more details about creating standard html forms for direct access.

Chapter 2—Omnis Web Client Tutorial

Developing an Omnis web application using the Omnis web client is relatively easy once you have learnt how to develop Omnis applications in general. The first part of this chapter describes how you can create a simple remote form to browse the Books data supplied in the Examples folder in Omnis Studio. It uses one of the remote form wizards as a starting point for your form and involves some programming. You can create the Books remote form and test it locally in the development version of Omnis. The second part of the tutorial describes how you can deploy the Book browser window, and assumes you have a personal web server such as the one supplied with Microsoft FrontPage, and it uses the development version of Omnis to run the application. However for the final deployment stage in your own Omnis web application, you will need to purchase and install a stand-alone web server, set up a Win32 or Linux server to run the Omnis runtime and your library, and you'll need to purchase the appropriate license for the Omnis runtime.

IMPORTANT: You must install the Omnis web client to do the tutorial, or design and test any remote form for that matter. There is an installer for the Omnis web client on the Omnis Studio CD in the Webclient/Client folder. There is a separate installer for the ActiveX (Windows only) suitable for displaying your forms in Microsoft Internet Explorer, and a Netscape plug-in suitable for Netscape Navigator. You can use either the ActiveX or Netscape plug-in for the tutorial, but you must run the appropriate installer before you begin the tutorial.

This tutorial assumes you know how to use the Omnis Component Store and Property Manager to create and modify fields and components, and it also assumes you know how to use the method editor to enter methods and variables. For further details about all these topics, see the *Using Omnis Studio* manual.

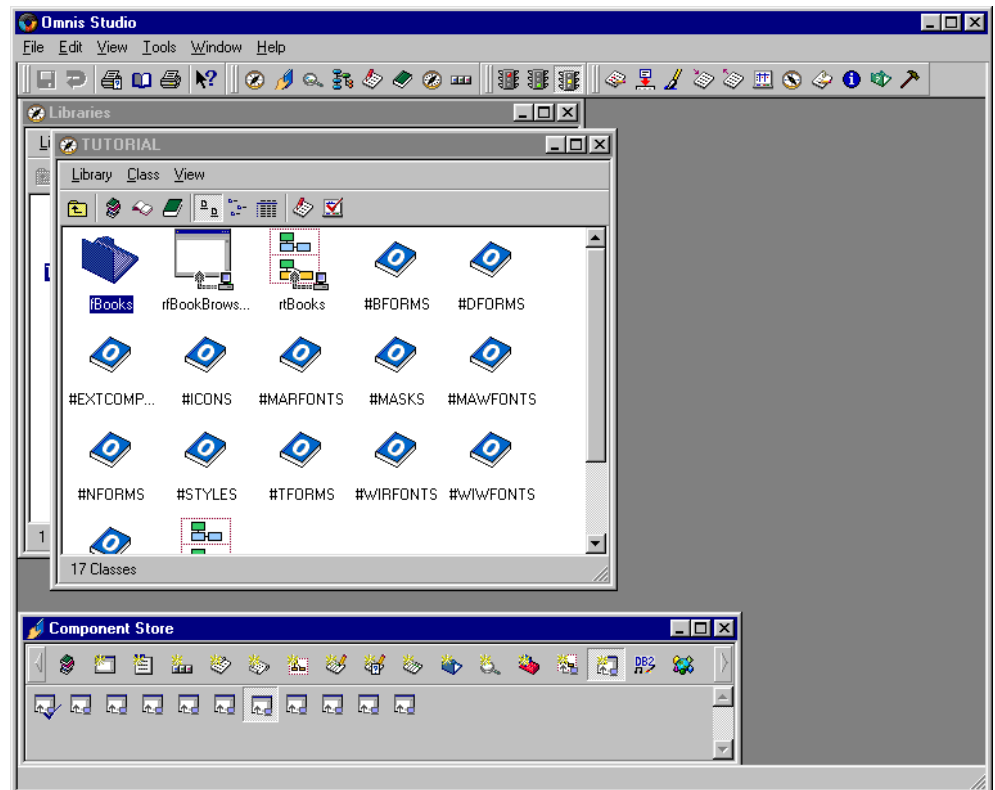
Designing the Remote Form

To develop an Omnis web application you need to create an Omnis remote form using one of the wizards supplied in Omnis. First, open the Tutorial library:

- Start Omnis and press Ctrl/Cmnd-O to open the library
- Open the Welcome folder in the main Omnis Studio folder
- Open the Tutorial/Webclient folder and double-click on the Tutorial.lbs library

The library opens a sample datafile called Webclient.df1 containing the Books data. If the datafile cannot be found, the library will prompt you to open the Webclient.df1 datafile, which is located in the Welcome/Tutorial/Webclient folder in your main Omnis Studio folder.

- Double-click on the library in the Class Browser to show all its classes

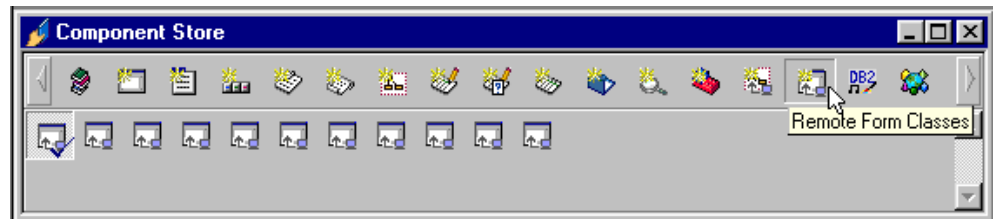


The library contains a number of classes including the **fBooks** file class, the **rtBooks** remote task class, and the **#ICONS** system table containing the necessary icons. The library also contains a remote form class called **rfBookBrowserFinal** which is a backup copy of the form you create in this tutorial. The **Startup_Task** class contains code that is executed when the library is opened, which in this case opens the Webclient.df1 example datafile.

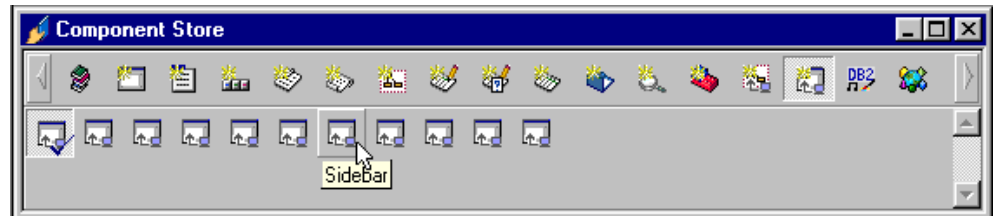
Using the Sidebar Remote Form Wizard

First you need to open the Component Store and view the remote form class wizards.

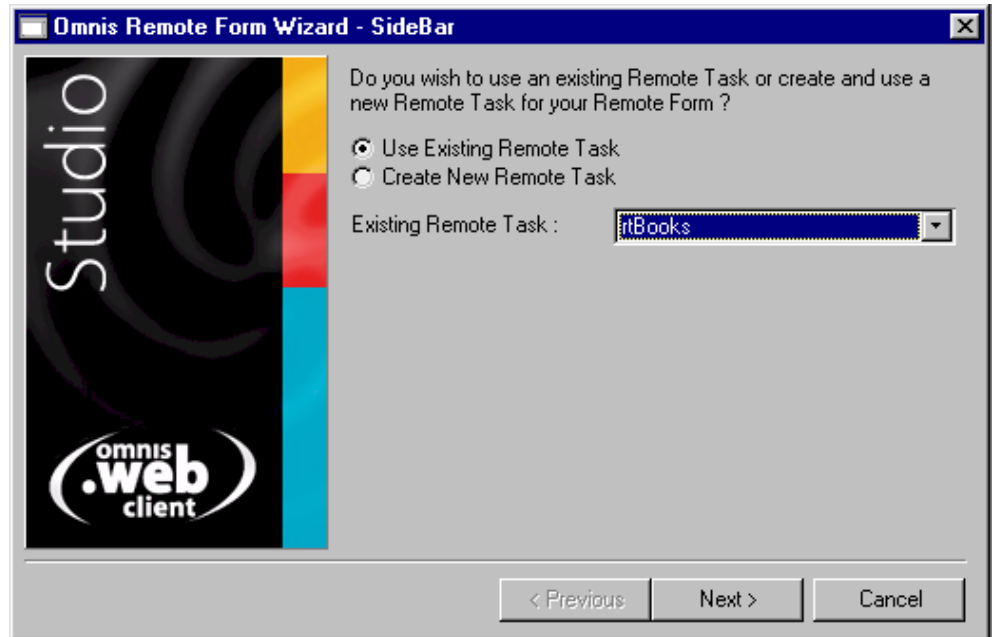
- Press F3/Cmnd-3 to open the Component Store or bring it to the top



- Click on the Remote Form Classes button to show the remote form wizards and locate the Sidebar wizard

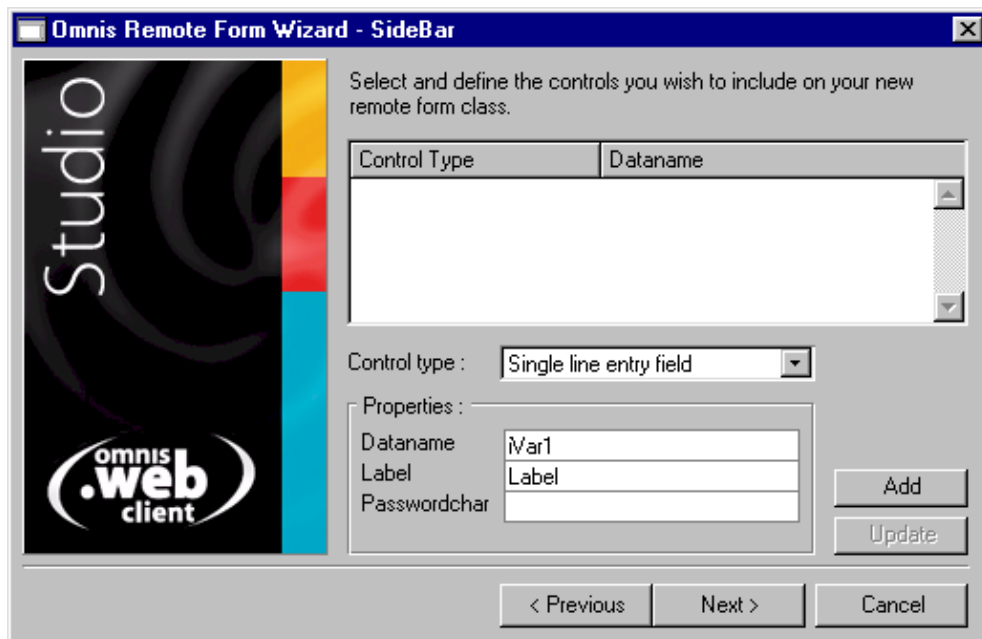


- Drag the **Sidebar** wizard onto the Tutorial library in the Class Browser (note do not use the 'Sidebar with pages' wizard)
- Name the class **rfBookBrowser** and press Return



The wizard prompts you for a remote task, or it allows you to create a new one. For this tutorial you can use the remote task supplied in the tutorial library.

- In the wizard, select **rtBooks** from the Existing Remote Tasks list, and click on the Next button



The wizard now lets you enter the details for the fields you require on your form. You can specify the component type, dataname, and text label for each field on your form.

For the first field

- Select **Heading List** as the component type
- Enter **iBookList** as the dataname
- Enter **3** as the number of columns
- Enter **Title,Author,Cost** (no spaces) for the Column titles
- Enter **200,120,50** as the column widths
- When the above details are correct press the **Add** button

For the second field

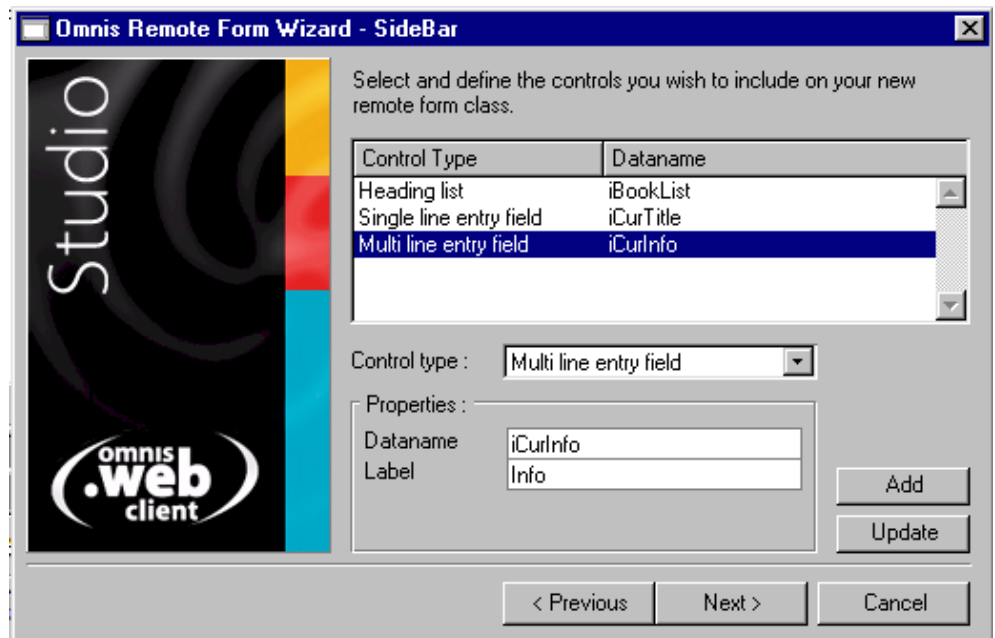
- Select **Single line entry field** as the component type
- Enter **iCurTitle** as the dataname
- Enter **Title** as the label

- When the details are correct press the **Add** button

For the third field

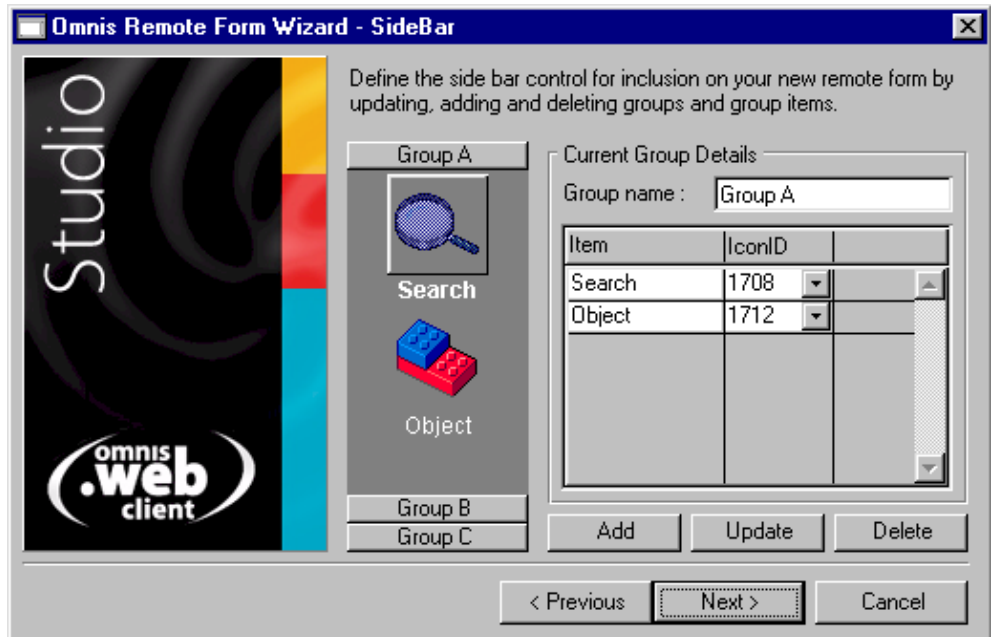
- Select **Multi line entry field** as the component type
- Enter **iCurInfo** as the dataname
- Enter **Info** as the label
- Press the **Add** button when the details are complete

The wizard window should look like this:



When you are sure these details are correct

- Click on the **Next** button in the wizard window



The wizard now lets you define the group and group items for the sidebar on your remote form. For our example remote form, you need to create a sidebar with 5 groups. The wizard has three groups which you can update, and you need to add two further groups.

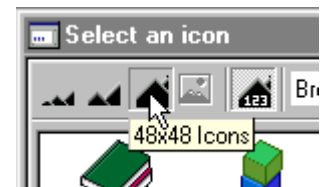
- In the Group name entry field replace the text Group A with **Bestsellers** and tab to the first item name in the list
- Replace the text with **Bestsellers** and tab to the Icon id cell

The icons for the Book browser sidebar are stored in the tutorial library in the #ICONS table, a system table stored in the library. You need to enter the Icon id of the icon for the Bestsellers group of books. To do this:

- Replace the current number in the IconID field with **1**

Alternatively, you can select the icon from the Select an Icon dialog. To do this:

- In the wizard window, click on the dropdown arrow in the IconID field to open the Select an Icon dialog
- Click on the 48x48 Icons button in the toolbar in the dialog
- Next, click on the top right-most button



marked #ICONS



- In the Select Icon dialog, click on the icon numbered **1** and press the **Select** button

Back in the wizard, you don't need the second item in the group so you can delete it using the context menu over the item.

- Right-button click (Windows) or Ctrl-click (Mac) on the second line in the list and select the Delete option
- Click on the **Update** button to update the details for the first group

The wizard window should look like this:



- In the preview sidebar click on the Group B bar
- In the Group name entry field replace the text Group B with **Art & Design** and tab to the first item name in the list
- Replace the text with **Art** and tab to the Icon id cell
- Replace the current number with **2** and tab to the second line in the list (or select the icon from the Select an Icon dialog as above)
- Right-button click (Windows) or Ctrl-click (Mac) on the second line in the list and select the Delete option
- Click on the **Update** button to update the details for the second group

Now the wizard should look like this:



- In the preview sidebar click on the Group C bar
- In the Group name entry field replace the text Group C with **Cooking** and tab to the first item name in the list
- Replace the text with **Cooking** and tab to the Icon id cell
- Replace the current number with **3**
- Tab to the second line in the list to edit it
- Replace the current text with **Food** and tab to the Icon id cell
- Replace the current number with **4**
- Click on the **Update** button to update the details for the third group



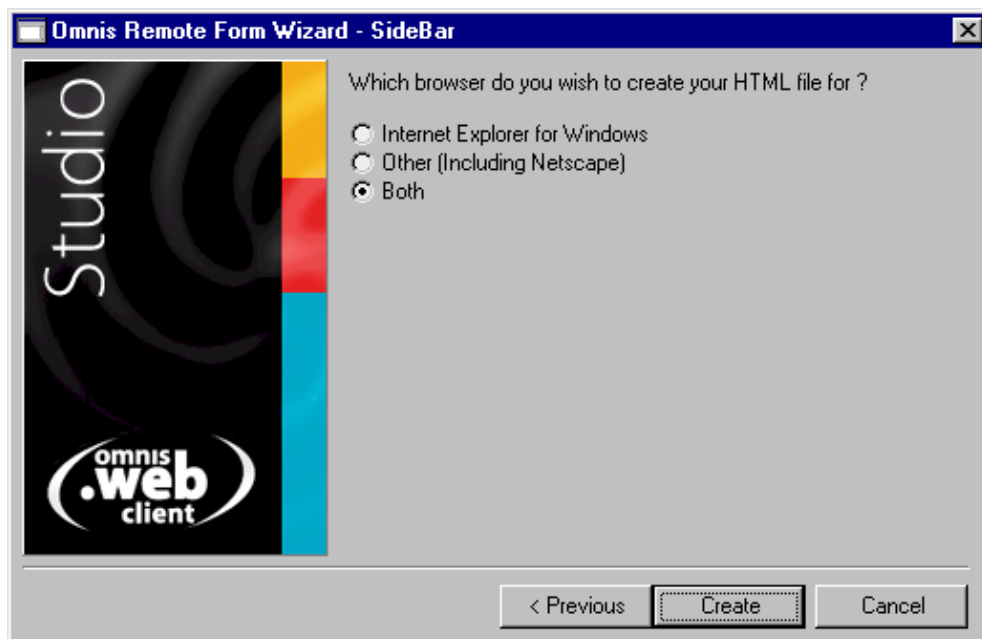
- In the Group name entry field replace the text **Cooking** with **Science** and tab to the first item name in the list
- Replace the text with **Science** and tab to the Icon id cell
- Replace the current number with **5** and tab to the second line in the list
- Replace the text with **Technology** and tab to the Icon id cell
- Replace the current number with **6**
- This time rather than updating, you need to click on the **Add** button to add the details for the fourth group



- To enter the fifth group, in the Group name entry field replace the text Science with **Fiction** and tab to the first item name in the list
- Replace the text with **Fiction** and tab to the Icon id cell
- Replace the current number with **7** and tab to the second line in the list
- Right-button click (Windows) or Ctrl-click (Mac) on the second line in the list and select the Delete option
- Again rather than updating, you need to click on the **Add** button to add the details for the fifth group



- When all the group details, item names and icon ids for each group item are correct, click on the **Next** button



The wizard now prompts you to choose the type of web browser you intend to use; this also depends on the type of Omnis web client you installed before starting the tutorial.

- Select **Internet Explorer** or **Netscape** and click on the **Create** button

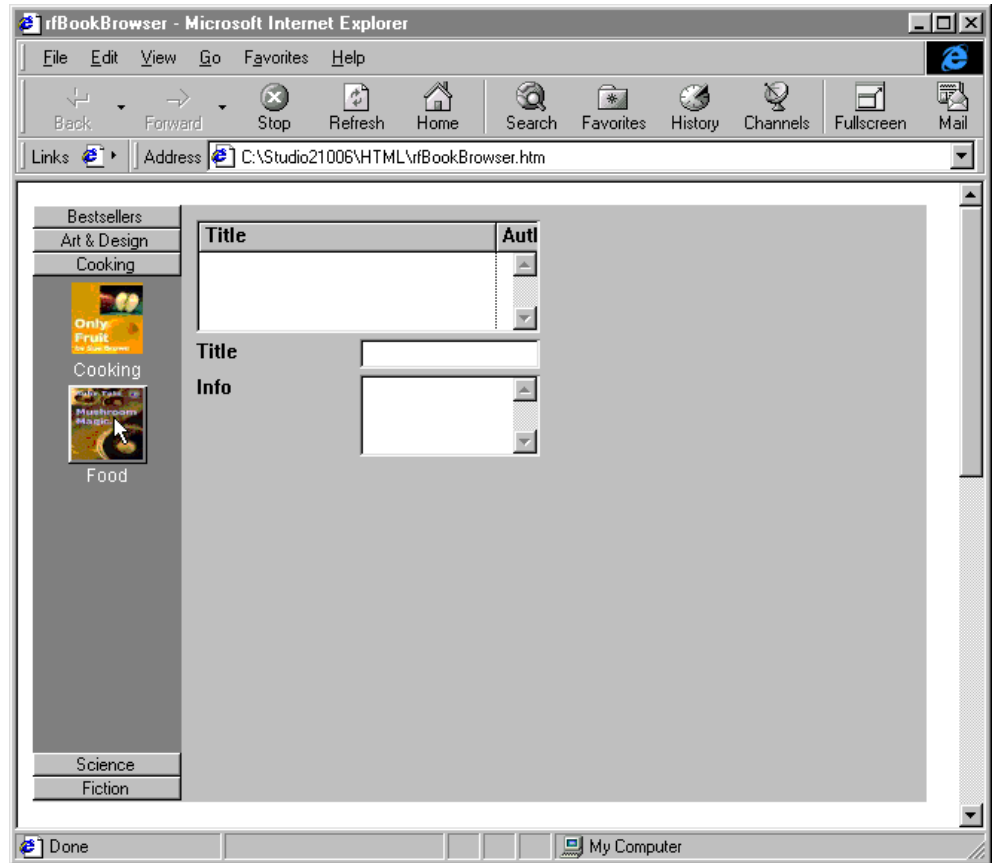
The **rfBookBrowser** remote form has been added to the tutorial library and should appear in the Class Browser. To view the form in design mode:

- Double-click on the **rfBookBrowser** remote form in the Class Browser

The screenshot shows a design window for a remote form titled "Remote Form TUTORIAL.rfBookBrowser". The window is divided into a left sidebar and a main content area. The sidebar contains three "IconName" labels and two buttons labeled "Mail" and "Other". A red arrow points to the sidebar with the label "Sidebar component". The main content area contains three fields: a "Title" field (labeled "Sidebar_1002"), another "Title" field (labeled "Sidebar_1004"), and an "Info" field (labeled "Sidebar_1006"). Red arrows point to these fields with labels: "Book list field" for the top field, "Book title field" for the middle field, and "Book info or details field" for the bottom field.

The form created by the Sidebar wizard contains the Sidebar component down the left-hand side and the three fields you specified, plus their text labels. The **BookList** (Heading List component type) is at the top which will hold the list of books, the **Title** field (single-line edit) is next which will display the current book title, and the **Info** field (multi-line edit) is at the bottom which will hold details about the currently selected book. Note that the Sidebar does not show the groups you entered, they will however appear when you view the form in your browser.

This form does not contain any programming to fetch and display the Books data, but you can open or "test" the form at any time by pressing Ctrl/Cmnd-T. Omnis opens your web browser automatically and displays the form.



At this stage your form does not handle the Books data, but this gives you an idea of how the form appears in a web browser. You can however click on the Sidebar and change book groups. Note that the browser uses a template html page created for you by the form wizard. The template page is placed in the **html** folder in the main Omnis folder. This html page has the Omnis web client embedded in it by default with all the correct parameters set up for you automatically.

If your web browser containing the remote form does not open at this point, you should check that you have installed the Omnis web client; if you installed the ActiveX check that it is registered properly. See the Troubleshooting guide for details about registering the web client engine manually.

- Close or minimize the web browser to return to Omnis; the remote form class should still be on top in Omnis in design mode

Editing Form Fields

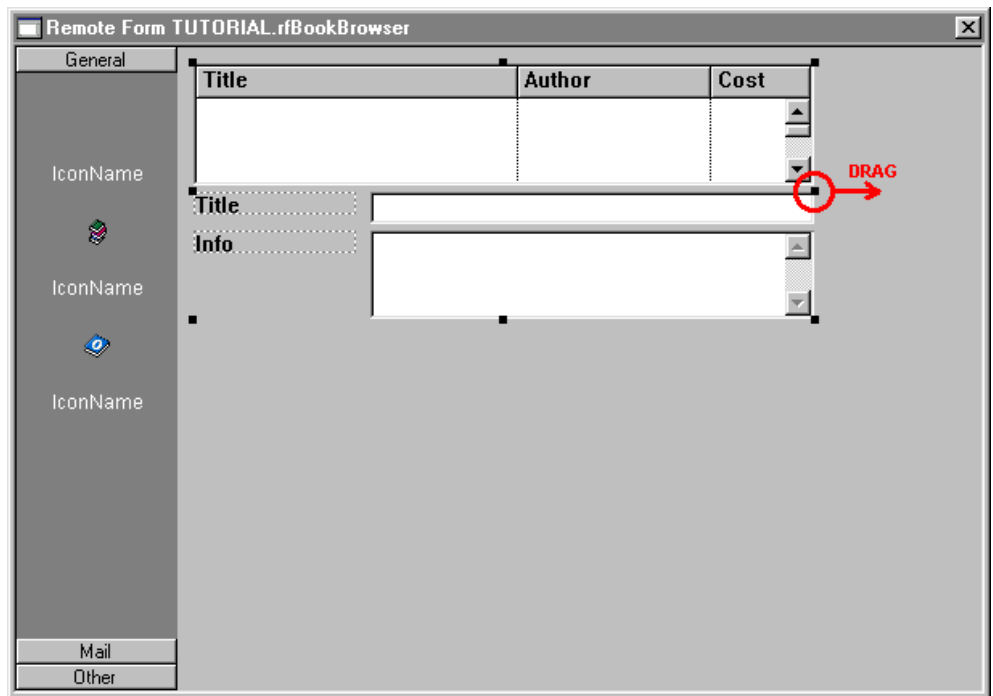
The next few steps involve resizing and renaming the three fields on the form, and adding some programming behind the remote form and, in particular, the BookList and Sidebar. Later you can add a picture field to show the book covers.

First, to resize the Book list, book title and info fields you can either resize each one in turn or select all three and resize them as a group.

- In the remote form class, click on each field and resize it; the **Cost** column should be visible in the Booklist and you should resize the title and info fields to match

Alternatively, to resize them as a group:

- Shift-click on the three fields to select them all
- Drag the right handle of the group selection to make all three fields wider, as shown below



- Click on the background of the form to deselect all the fields
- At this stage you may like to make the Info field deeper by dragging its bottom handle

- Next, click on the BookList at the top and press F6/Cmnd-6 to open the Property Manager, or bring it to the top
- On the General tab in the Property Manager change the **name** property to **BookList**

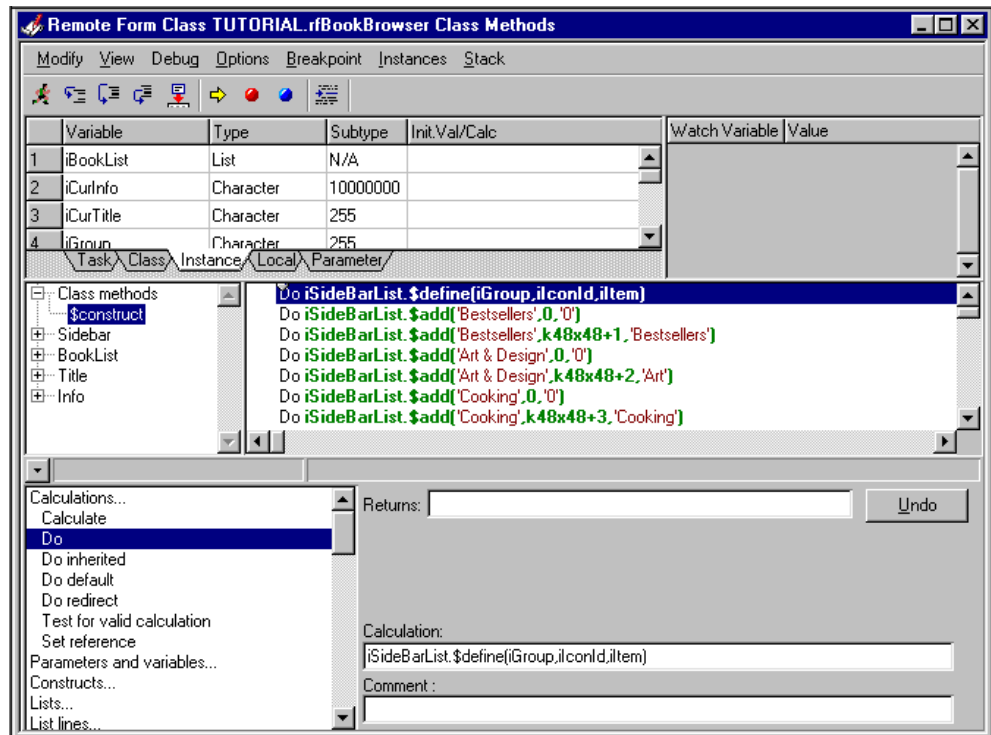


- Similarly, click on the Title field (the single line field), open the Property Manager, and change its name to **Title**
- Then click on the Info field (the multi-line one), open the Property Manager, and change its name to **Info**

Adding Variables and Methods to the Form

Having altered the fields, you must add some programming behind the form and its components. You need to add some variables to the form, edit some of the code that the wizard created for you, and add one or two new methods.

- Double-click on the background of the form, in design mode, to open the method editor for the remote form class



- Click on the **Instance** tab in the variable pane
- Right-click/Ctrl-click on the variable pane and select **Insert New Variable** from the context menu
- Name the variable **iBookGroup** and select Type **Number** and Subtype **Long Integer** from the appropriate droplists
- To insert another variable, Right-click/Ctrl-click on the variable pane and select **Insert New Variable** from the context menu
- Name the variable **iCurCover** and select **Picture** from the Type droplist (it doesn't require a subtype)

You should now have 9 variables in your remote form. Next you are going to edit the code in the \$construct method, which should be selected in the method editor. This method builds the group items and icon details for the Sidebar component in the form. First, edit the first line of code:

```
Do iSideBarList.$define(iGroup,iIconId,iItem)
```

by adding the **iBookGroup** variable to the list definition. The line should now be:

```
Do iSideBarList.$define(iGroup,iIconId,iItem,iBookGroup)
```

Next you need to add the book groups or categories to each line of code that defines each one of the group items in the Sidebar component. The following table gives you the categories (the same as stored in the Books data file).

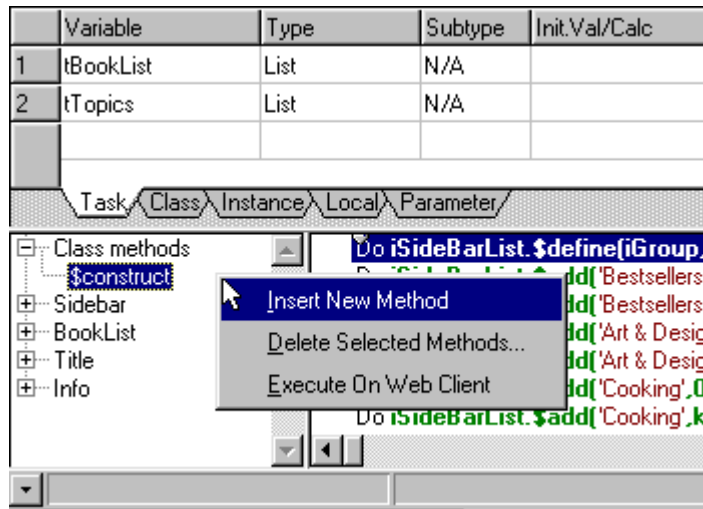
Bestsellers	0
Art	1
Cooking	2
Food	3
Science	4
Technology	5
Fiction	6

The lines you need to edit are shown highlighted (the additions are underlined and in red), and shows you the final method. If you prefer, you can copy the method from the on-line PDF manual and Paste it into the method editor replacing the existing code, or you can copy the complete method from the rfBookBrowserFinal class in the tutorial library.

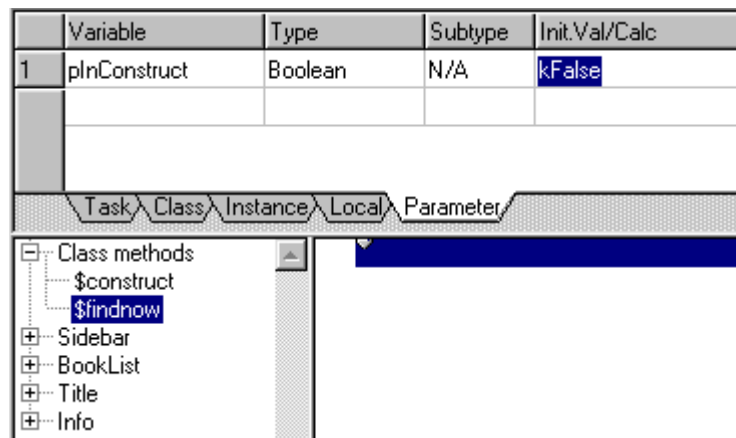
```
Do iSideBarList.$define(iGroup,iIconId,iItem,iBookGroup)
Do iSideBarList.$add('Bestsellers',0,'0')
Do iSideBarList.$add('Bestsellers',k48x48+1,'Bestsellers',0)
Do iSideBarList.$add('Art & Design',0,'0')
Do iSideBarList.$add('Art & Design',k48x48+2,'Art',1)
Do iSideBarList.$add('Cooking',0,'0')
Do iSideBarList.$add('Cooking',k48x48+3,'Cooking',2)
Do iSideBarList.$add('Cooking',k48x48+4,'Food',3)
Do iSideBarList.$add('Science',0,'0')
Do iSideBarList.$add('Science',k48x48+5,'Science',4)
Do iSideBarList.$add('Science',k48x48+6,'Technology',5)
Do iSideBarList.$add('Fiction',0,'0')
Do iSideBarList.$add('Fiction',k48x48+7,'Fiction',6)
```

You can save the form at any time by Selecting the **File>>Save rfBookBrowser** from the main Omnis menubar. Next you are going to add a method to the remote form.

- In the method editor, Right-click/Ctrl-click on the Class methods item in the method editor, and select **Insert New Method** from the context menu



- Name the new method **\$findnow**
- Click on the **Parameter** tab in the variable pane and add a new variable by Right-clicking/Ctrl-clicking in the variable list
- Name the new variable **pInConstruct**, select **Boolean** from the Type droplist, and give it an Initial value of **kFalse**



- Click in the method editor again and enter the following code for the **\$findnow** method

Again, to save typing in this method you can copy the code from the on-line PDF manual and paste it into the \$findnow method, or you can copy and paste the complete method from

the `rfBookBrowserFinal` class in the tutorial library. Some lines may be commented out, but try un-commenting these lines individually.

```
Set main file {fBooks}  
Set current list iBookList  
Load from list  
Find on fBooks.BookTitle (Exact match)  
Calculate iCurCover as fBooks.BookCover  
Calculate iCurInfo as fBooks.BookInfo  
Calculate iCurTitle as fBooks.BookTitle
```

```
Do $cinst.$objs.Info.$redraw()  
Do $cinst.$objs.Cover.$redraw()  
Do $cinst.$objs.Title.$redraw()
```

```
If not(pInConstruct)  
    Do $cinst.$senddata(iCurCover,iCurInfo,iCurTitle)  
End If
```

Next you need to add some code to the `$construct()` method to call the `$findnow()` method.

- Click on the **\$construct** method and scroll to the bottom of the lines of code in the method
- Add the following lines of code; if you prefer, try Copying and Pasting the code as before

```
; Build book list  
Calculate iBookList as tBookList  
Set current list iBookList  
Calculate #L as 1  
Do method $findnow (kTrue)
```

The complete \$construct method, including all the edits and additions, is shown below; if you have difficulty with this method, you can copy the complete method from the rfBookBrowserFinal class in the tutorial library.

```
Do iSideBarList.$define(iGroup,iIconId,iItem,iBookGroup)
Do iSideBarList.$add('Bestsellers',0,'0')
Do iSideBarList.$add('Bestsellers',k48x48+1,'Bestsellers',0)
Do iSideBarList.$add('Art & Design',0,'0')
Do iSideBarList.$add('Art & Design',k48x48+2,'Art',1)
Do iSideBarList.$add('Cooking',0,'0')
Do iSideBarList.$add('Cooking',k48x48+3,'Cooking',2)
Do iSideBarList.$add('Cooking',k48x48+4,'Food',3)
Do iSideBarList.$add('Science',0,'0')
Do iSideBarList.$add('Science',k48x48+5,'Science',4)
Do iSideBarList.$add('Science',k48x48+6,'Technology',5)
Do iSideBarList.$add('Fiction',0,'0')
Do iSideBarList.$add('Fiction',k48x48+7,'Fiction',6)

; Build book list
Calculate iBookList as tBookList
Set current list iBookList
Calculate #L as 1
Do method $findnow (kTrue)
```

- When the \$construct method is complete, close the method editor

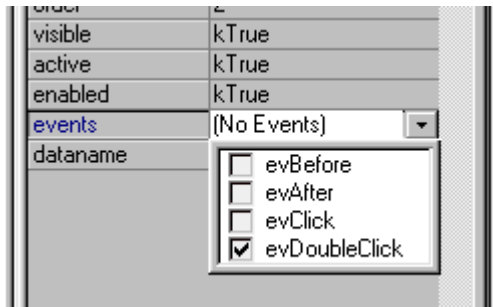
At this stage, you can save the changes you have made to the form by selecting the **File>>Save rfBookBrowser** from the main Omnis menubar. You still have to add some code behind the BookList and Sidebar, but you may like to try the form in your web browser again. When you open the form this time, using Ctrl/Cmnd-T, the BookList should contain the books from the Bestsellers category, but at this stage clicks on the Sidebar and the BookList won't have any effect.

- To return to Omnis, close or minimize your web browser

Events and Adding Methods to Form Fields

Next you are going to add some code behind the BookList. First though you need to enable the `evDoubleClick` event for the list. Normally, all field events are enabled for window objects, but for web components you need to explicitly enable which events you want to be processed by the object. You can do this by setting the object's **events** property.

- In the remote form, click on the **BookList** field and press F6/Cmnd-6 to open the Property Manager or bring it to the top
- Under the General tab, click on the **events** property and select/check the **evDoubleClick** event in the droplist



Now you need to add the method for the BookList.

- In the remote form, double-click on the **BookList** field to open the method editor for the object
- Enter the following code in the \$event method

```
On evDoubleClick  
Do method $findnow
```

- When you've entered the method, close the method editor window

Next you are going to enter the `$event()` method for the Sidebar component. This time however, you do not need to enable any events for the Sidebar, that is `evIconPicked`, since the Sidebar wizard has done this for you automatically. To enter the event handler for the Sidebar

- Double-click on the **Sidebar** component in the form

- In the method editor, enter the following code in the \$event method

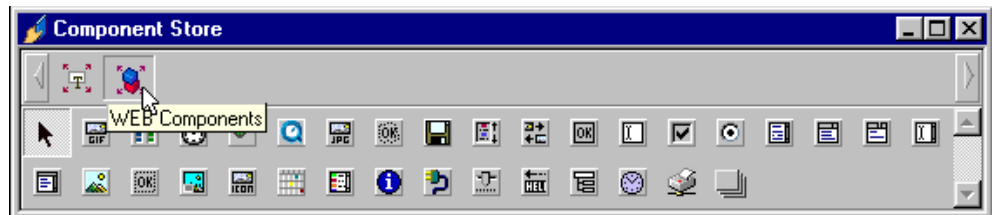
```
On evIconPicked
  Set current list iSideBarList
  Load from list {pLinenum}
  Do method $ctask.$buildbooks (iBookGroup)
  Calculate iBookList as tBookList
  Do $cinst.$objs.BookList.$redraw()
  Do $cinst.$senddata(iBookList)
```

- When you've entered the method, close the method editor window, and save the class using **File>>Save rfBookBrowser**

Adding a Picture Field

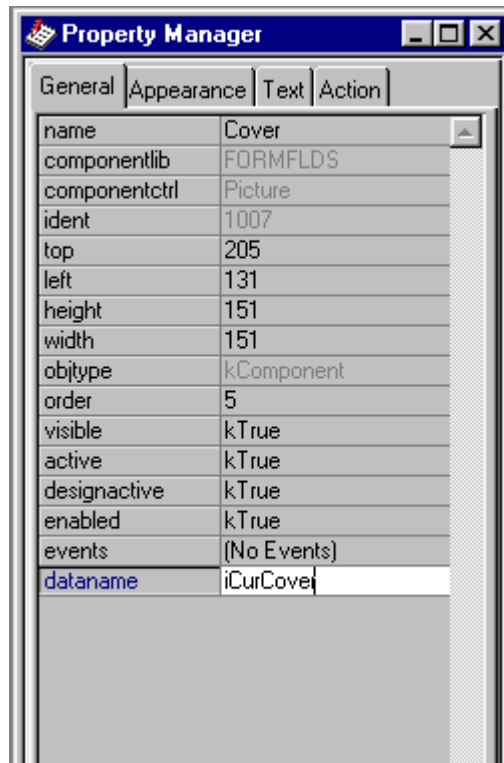
And finally you are going to add a picture field to the form to view the book covers. You can create picture fields in the Sidebar wizard, but this section lets you add a field to your remote form using the Component Store.

- Assuming the form is on top, press F3/Cmnd-3 to open the Component Store or bring it to the top, and click on the **WEB Components** button in the Component Store toolbar



The Component Store contains over 30 different web components including sidebar, progress bar, list components, form fields, buttons, check boxes, Quicktime movie player, and so on. For the tutorial, you need to locate the Picture component by moving your mouse over the buttons to view the help tips.

- Drag the **Picture** object onto your form and release the mouse
- With the Picture selected, press F6/Cmnd-6 to open the Property Manager, or bring it to the top
- Under the General tab, change the **name** property to **Cover**
- Enter the **dataname** (\$dataname) as **iCurCover**



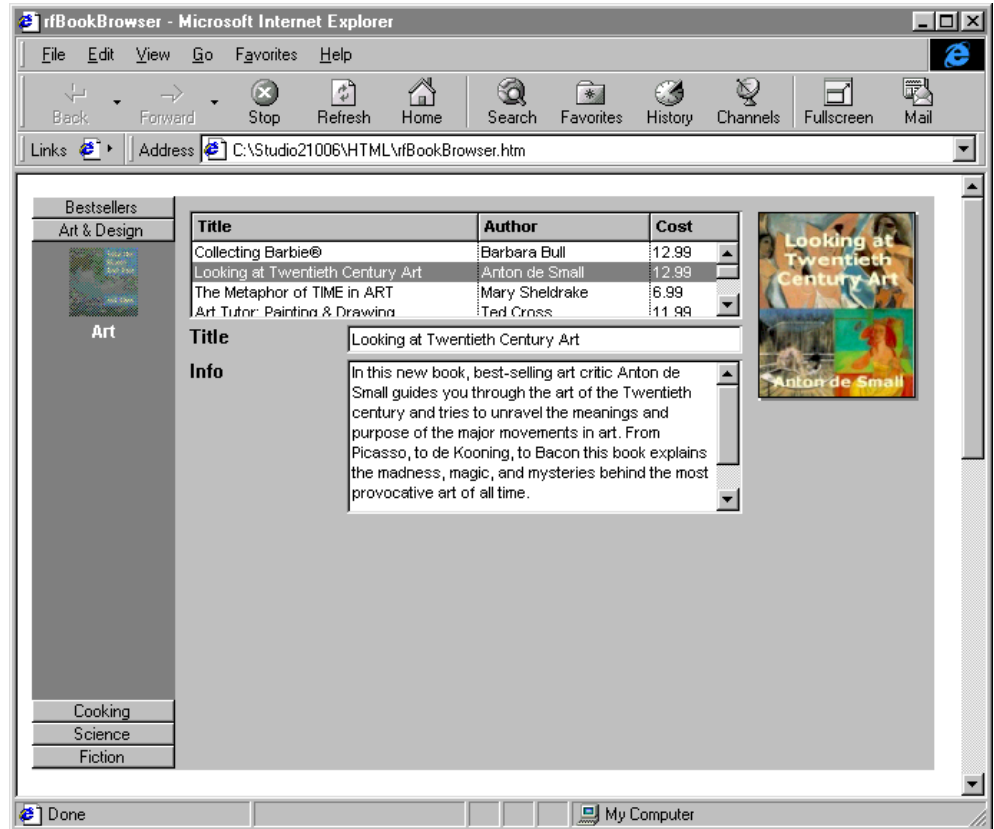
You may like to remove the scroll bars from the picture field, and resize and position it in the form. To do this:

- Click on the Picture field and press F6/Cmnd-6 to open the Property Manager, or bring it to the top
- Under the Appearance tab, set both the properties **vertscroll** and **horzscroll** to **kFalse**
- Under the Appearance tab, set the **effect** property to **kShadowBorder**
- Under the General tab, set the **height** property to **132**, and the **width** property to **112**
- Make the form a bit wider and move the picture field up to the top right-hand corner of the form, to the right of the BookList field

Testing the Form and Browsing the Books

To test the remote form:

- Press Ctrl/Cmnd-T to try out the form, and browse the Books data



This form allows you to browse the book details. You can click on the Sidebar to change the group or subgroup to view the different categories of books, and you can **double-click** on the BookList to view the details and cover for individual books.

Debugging the Form

When you are developing your remote form you can set breakpoints in your code in the method editor, and use trace mode. If you set a breakpoint in one of the methods, open the remote form in your web browser (using Ctrl/Cmnd-T) and use the form, Omnis will switch back to the debugger in design mode. Similarly, if trace mode is enabled Omnis will trace all the methods that are called in the form, allowing you to see what's going on.

Deploying the Remote Form

To deploy an Omnis web application using the Omnis web client you need a Win32 or Linux web server, and a Win32 or Linux server to install the runtime version of the Omnis executable. However, for the purposes of this tutorial you can use a Personal web server set up in Microsoft FrontPage, or Apache under Linux, and the development version of Omnis on your local machine. The deployment process is exactly the same in both cases, which involves:

- ☐ installing and configuring the Omnis Server, that is, the Omnis runtime and your Omnis library
- ☐ installing and configuring your web server, and editing your html pages

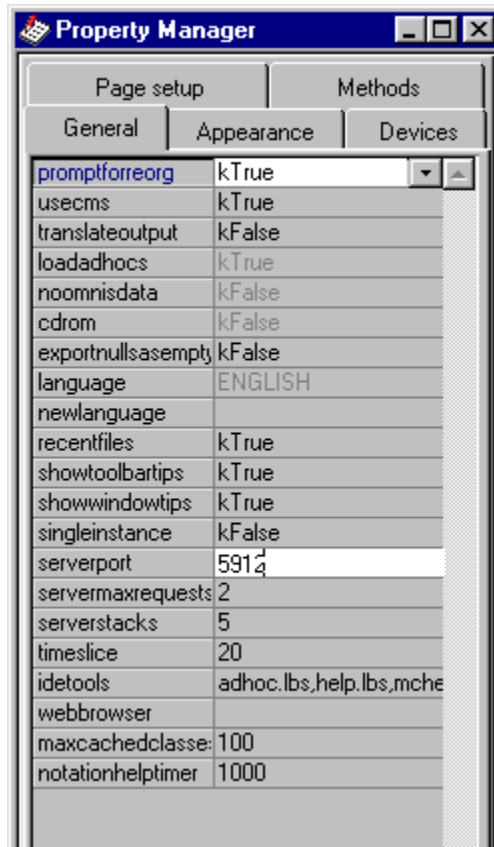
If you go ahead and deploy the tutorial library to your web server, rather than just using it locally, you must edit the `$construct()` method in the `Startup_Task`. In particular you need to edit the `Open data file` command and associated code that opens the `Webclient` datafile, and you must remove the *Prompt for data file* command. You must not use any commands in the Omnis application that display a prompt of the server.

Omnis Server

The tutorial uses the development version of Omnis, but if you are deploying the library to the web you must install and serialize the Omnis runtime.

For the Omnis development version:

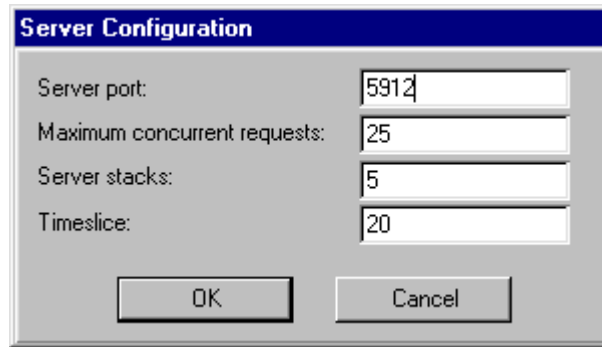
- Start Omnis, open the Tutorial library and connect to the Books data file
- Select **Tools>>Options** from the main Omnis menubar



- In the Omnis preferences in the Property Manager, set the **serverport** property to 5912, or if this number is already in use, choose some other number
- Restart Omnis, reopen the library and keep it running in the background

For the Omnis Server runtime:

- Install the Omnis runtime onto your Win32 or Linux server and serialize it using a web serial number
- Copy the tutorial library and the datafile containing the Bookstore data (Webclient.df1) to your Win32 or Linux server
- Start the Omnis runtime
- Select **File>>Server Configuration** from the main Omnis menubar



- Enter 5912 in the Server port field and click OK

The port number must be available on both the Omnis runtime and the web server. The number can be between 1 and 32767, but you must not use 80 or any other number used for e-mail, FTP, or other services.

- Open the Tutorial library and connect to the Books data file; keep the library running in the background

Web Server and HTML files

The tutorial uses a Personal web server set up using Microsoft FrontPage, but if you are deploying the application to the web you must purchase and install a standard Win32 or Linux web server. You may be able to get a free or trial version of a web server from the Internet, or purchase a basic web server suitable for testing purposes. In either case the web server configuration is the same.

- Install your web server, or set up a personal web server in Microsoft FrontPage; see the Help in Microsoft FrontPage for details about creating a new FrontPage web
- Go to your Omnis development tree and locate the Omnis/WebClient/Server folder
- Copy the Omnisapi.dll file to your web server **cgi-bin** folder
- Next, locate the **Html** folder in the main Omnis folder
- Copy the **rfBookBrowser.htm** file to your personal or local web server
- Open the **rfBookBrowser.htm** file in a text editor or Html source editor
- In the Html source for the rfBookBrowser.htm file, edit the following parameters, in particular you need to edit the WebServerURL and WebServerScript parameters

For Internet Explorer you need to edit the following parameters for the ActiveX:

```
<param name="OmnisServer" value="5912"> ;; the port number
<param name="OmnisLibrary" value="TUTORIAL">
; the library name less the file extension
<param name="OmnisClass" value="rfBookBrowser">
; the remote form name
<param name="WebServerURL" value="http://www.yourdomain.com">
; the domain name or IP address of your web server or
; personal web server
<param name="WebServerScript" value="/cgi-bin/omnisapi.dll">
; the location of the omnis web server plug-in
```

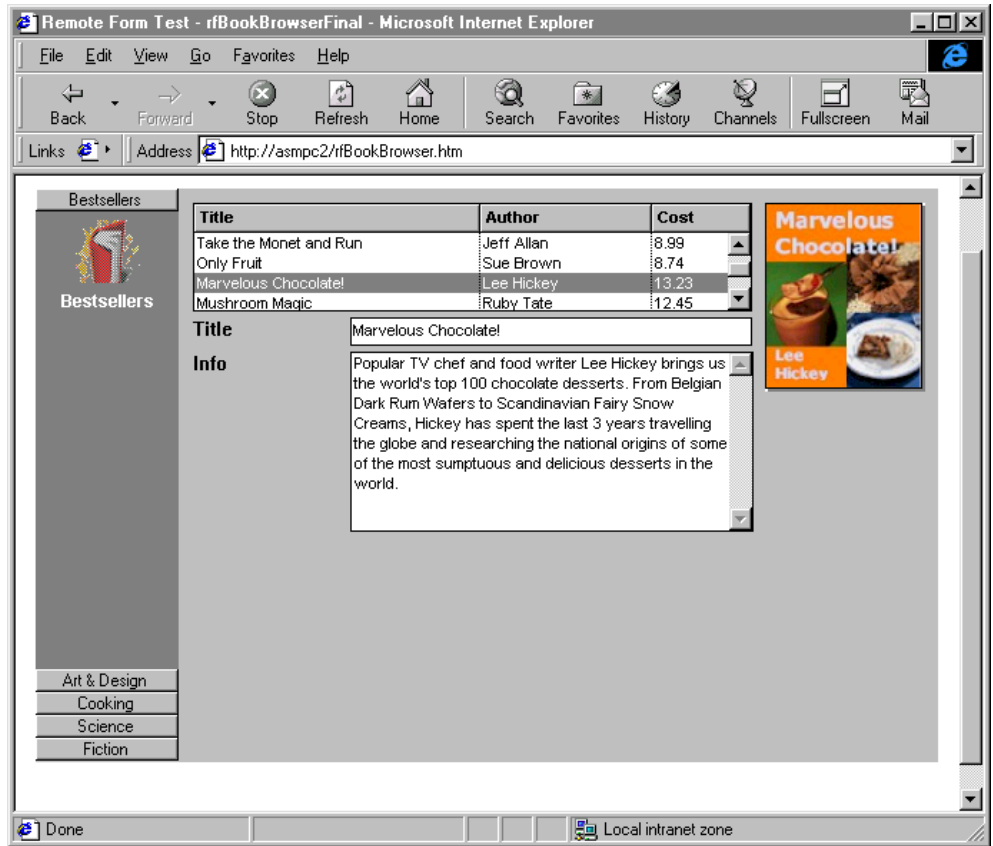
The parameters for the Netscape plug-in should be as follows:

```
<EMBED type=application/Omnis-RCC-plug-in name="rcc1"
width=500 height=500
OmnisServer="5912" ;; the port number
OmnisLibrary="TUTORIAL"
; the library name less the file extension
OmnisClass="rfBookBrowser"
; the remote form name
WebServerURL="http://www.yourdomain.com" ; the domain name or IP
address of your web server or personal web server
WebServerScript="/cgi-bin/omnisapi.dll" ; the location of the
omnis web server plug-in
Param1="pOption1=Data" .. .. Param9="pOption9=Data">
```

The port number, library name, and remote form name will already be set up for you, but if they're not, enter the values shown above. If your web server does not support the Microsoft ISAPI interface you will have to use the `nph-omnisapi.exe` cgi program or the Apache module. In this case, you need to add the `nph-omnisapi.exe` program to your `cgi-bin` folder and use this name in the **WebServerScript** parameter in your HTML file(s).

- Save and close the `rfBookBrowser.htm` file
- If you are deploying to a remote web server, copy the `rfBookBrowser.htm` file to your web server using FTP
- Open your web browser and navigate to the `rfBookBrowser.htm` file using "[YourDomainName or IPAddress]/rfBookBrowser.htm", where YourDomainName could be `http://www.yourdomain.com`, or IPAddress might be the IP address of your personal web server

The Book Browser form should appear and you can start to browse the books data.



Viewing the Omnis Web Client Example Library

You can see an enhanced version of the Books form in the Omnis web client example library available in the Welcome Application. The enhancements include a shopping basket to store the books you want to buy, and a checkout facility for entering credit card details. You can also view the Bookstore application on the Internet via the Omnis website www.omnis.net.

Chapter 3—Remote Forms

To create an Omnis web application you need to create a library containing your Omnis remote form classes and remote task classes. You can create and test the remote forms on your own machine using your own web browser, that is, you can develop your Omnis library before installing your web server, the web server plug-in and Omnis Server, or before setting up your html pages; these are described later in this manual.

General information about developing Omnis applications, designing forms and programming in Omnis, are found in the *Using Omnis Studio* and *Omnis Programming* manuals available with Omnis Studio.

You can create Omnis remote forms from scratch or using the templates and wizards available in the Component Store. The wizards give you complete control over the fields and components contained in your form and save you a lot of time. All the wizards create a new remote task or use an existing task to link to the new remote form.

Creating Remote Forms using Wizards

You can create a remote form using one of the wizards available in the Component Store. The wizards let you specify the fields and components you want on your form. The wizards are summarized here and described in more detail below.

- ☐ **New Remote Form**
creates a new blank remote form class containing no web components and without a remote task class; if you create a new form using this template you need to create your own remote task class and set the \$designtaskname property of the remote form yourself
- ☐ **File Form**
creates a remote form based on an Omnis file class; the wizard creates a field for every data field in your file class and provides a Next and Previous button to view your data
- ☐ **Multiform**
lets you create a complete website containing home page, navigation bar, and any number of remote forms created using any of the other wizards
- ☐ **Password**
creates a standard password form containing Password and Username fields; creates an instance variable for each form field
- ☐ **Plain**
creates a blank form and links it to a remote task; you must add your own instance variables and web components to this form

- ❑ **Report**
creates a remote form that allows you to print a report to Html; the wizard prompts you for an existing report class or it can create an example report
- ❑ **Sidebar with pages**
creates a form containing a page pane and a sidebar component; you specify the fields for each pane and link each pane to a particular group in the sidebar; creates an instance variable for each form field
- ❑ **Sidebar**
creates a form containing a number of fields and a sidebar component; creates an instance variable for each form field
- ❑ **Submit**
creates a standard submit form containing your fields, Submit and Clear buttons; creates an instance variable for each form field
- ❑ **Tabs**
creates a remote form containing a tab strip and page pane component and fields on each pane; creates an instance variable for each form field
- ❑ **Wizard**
creates a remote form containing a page pane component, fields on each pane, and Next, Previous, and Finish buttons; creates an instance variable for each form field

All the wizards prompt you for a remote task class: a *remote task* instance manages the connection to the client, and is a container for all the instances on a particular client. You can select an existing remote task in your library, or you create one based on the templates provided. For most types of remote form, you can select the Plain Remote Task template.

All wizards, except the Plain one, generate an html template which is placed in the **html** folder of the Omnis root. This html page lets you test the form in design mode using Ctrl/Cmnd-T and provides a template for your final html pages for your website. Each wizard prompts you to select the type of browser you want to create html pages for. You can select Internet Explorer or Netscape, or both, and in all cases the appropriate plug-in is embedded into your template html pages.

All wizards that create forms containing data fields or components prompt you for the **dataname** and component type of each field you require on your form. An instance variable is created for every field you add to a form.

All remote forms created using a wizard have a \$construct() method containing a parameter variable called *pParams* of type *Row Variable*. This row variable contains the parameters in the remote form.

Having created your remote form(s) using the Remote Form Wizards, you can modify them or add instance variables and web components, as appropriate, to complete your remote forms. The remainder of this chapter describes the different form wizards and web components available in Omnis.

Plain Form Wizard

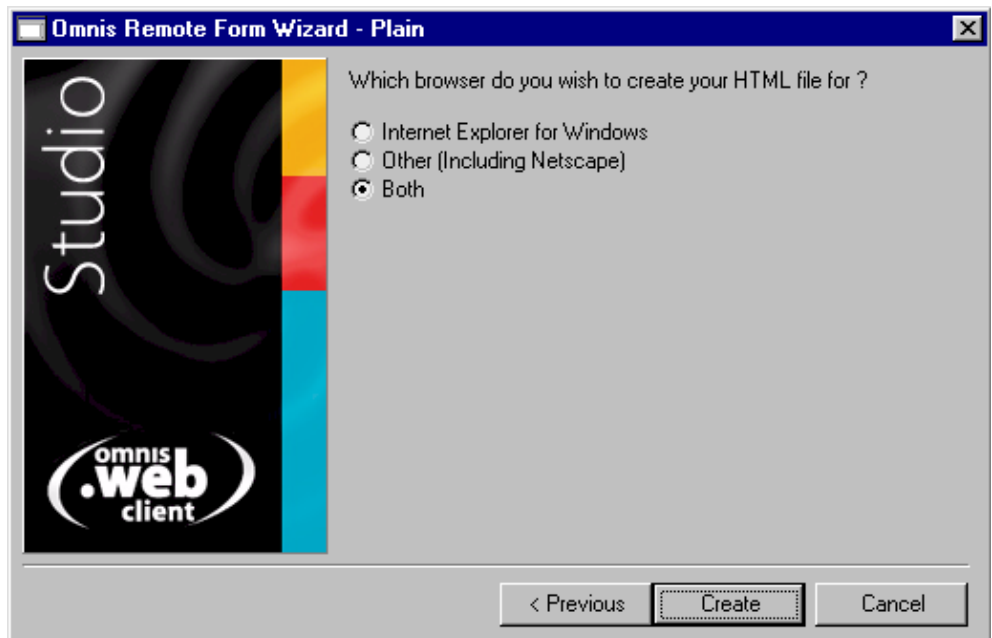
The Plain form wizard creates a blank form and links it to a remote task. The wizard prompts you for an existing remote task or lets you create a new one based on one of the templates provided. You need to add your own instance variables and web components to this type of remote form.

To create a Plain Form

- Open your library and display its classes in the Class Browser
- Open the Component Store and click on the Remote Form Classes button
- Select the Plain remote form wizard and drag its icon onto your library in the Omnis Class Browser
- Name the new remote form and press Return
- Select an existing remote task from the list of tasks provided, and click on the Next button

or

- Click on the Create New Task option, select a template from the list provided, name the new remote task, and click on the Next button



- When you have selected your browser option, click on the Create button

Omnis creates a new remote form in your library. Double-click on the form in the Class Browser to modify it.

Tabs Form Wizard

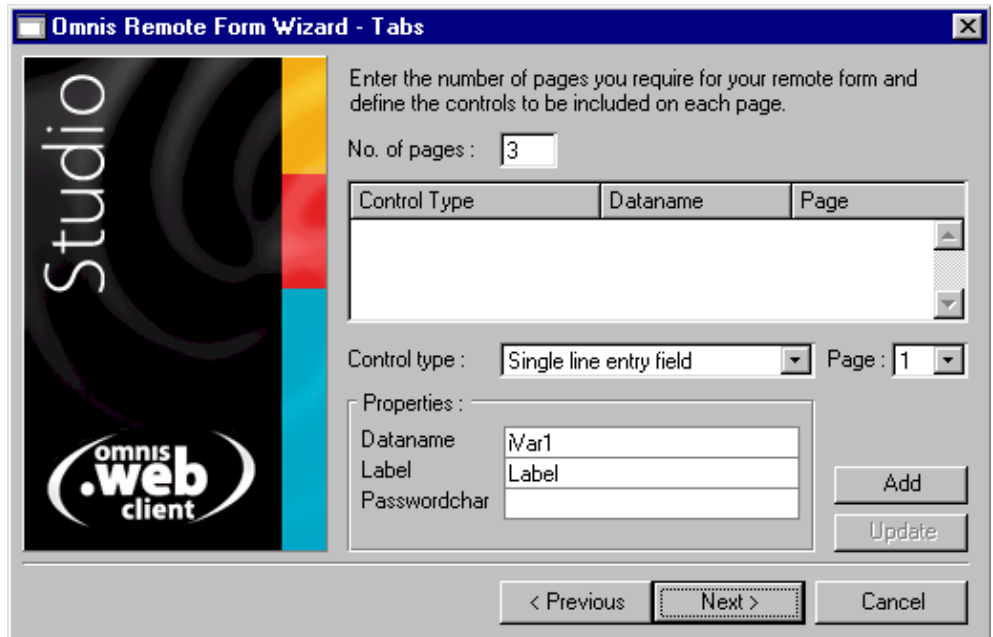
The Tabs form wizard creates a remote form containing a tab strip and page pane component. In the wizard you can add fields and components to each pane in the form. The resulting form lets the user enter data on a number of panes by clicking the tabs to change page.

To create a Tabs Form

- Open your library and display its classes in the Class Browser
- Open the Component Store and click on the Remote Form Classes button
- Select the Tabs remote form wizard and drag its icon onto your open library in the Omnis Class Browser
- Name the new remote form and press Return
- Select an existing remote task from the list of tasks provided, and click on the Next button

or

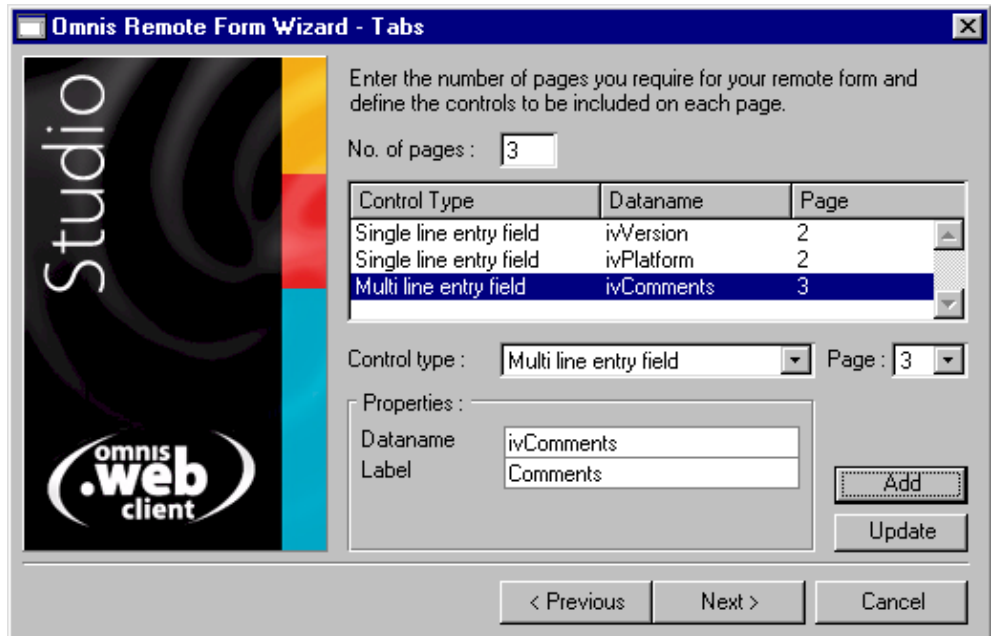
- Click on the Create New Task option, select a template from the list provided such as the Plain Remote Task, name the new remote task, and click on the **Next** button



- Enter the number of pages for your form
- Select the component type for the first field on the first page
- Enter the dataname and text label for the first field and click on the Add button
- Select the component type, enter the dataname and text label for the second field on the first page, and click on the Add button
- Add the remaining fields for the first page, then select the second page and add fields as above

The list in the wizard window shows the fields you have added to each page in the form. At any stage, you can click on a field in the list, edit its details, and click on the Update button.

The following screenshot shows a wizard that defines 3 pages, with a few fields on the first two pages, and a single multi-line field on the third page.

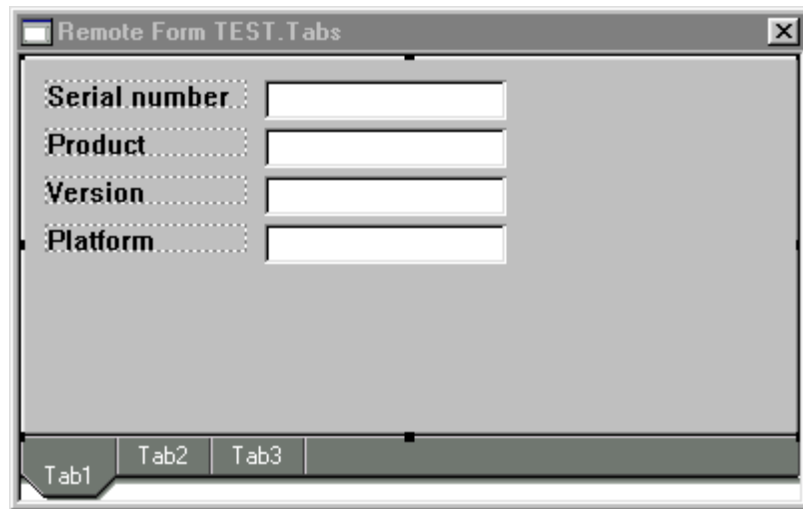


- When the fields for each page in your tabbed form are complete, click on the **Next** button
- Select the type of web browser and click on the **Create** button

The wizard now creates the remote form using the details you have entered. Note that Omnis creates a remote task as well.

- To modify the form, double-click on it in the Class Browser

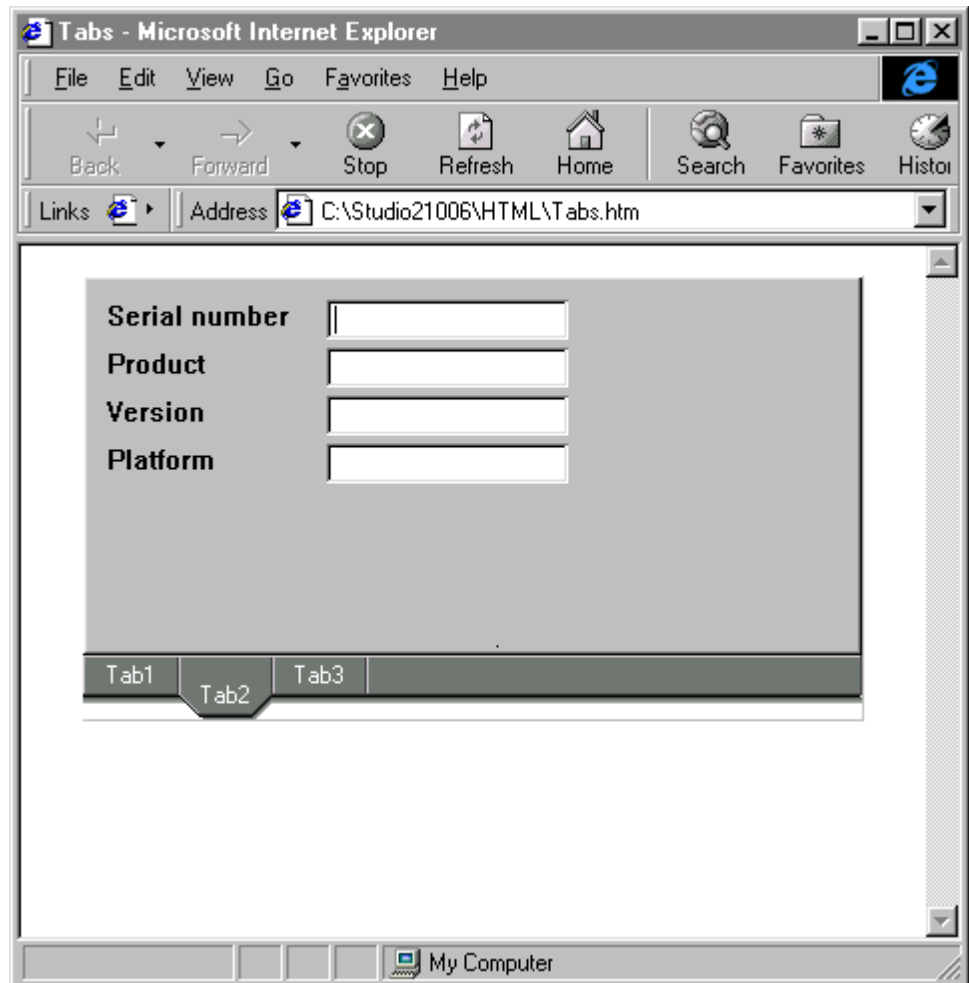
To examine the contents of each page in design mode, you need to click on the page pane field (click on the gray background, away from the fields and text labels), open the Property Manager or bring it to the top by pressing F6/Cmnd-6, and edit the **currentpage** property. For example, to look at the second page, click on the page pane field, enter 2 in the **currentpage** property in the Property Manager, and press Return (note the tab strip at the bottom of the window will not change in design mode). You should see something like the following screenshot.



To test the form

- Press Ctrl/Cmnd-T, or Right-click/Ctrl-click on the form and select the **Test Form** option from the context menu

When you press Ctrl/Cmnd-T, Omnis opens your browser automatically and displays the form.



The form is opened in your web browser and you can click on the tabs to change the current page on the form. At this stage your form does not handle the data you type into the form, but this gives you an idea of how the form will appear to the end user.

Note that the browser uses a template html page created for you by the form wizard. The template page is placed in the **html** folder in the main Omnis folder. This html page has the Omnis web client embedded in it by default with all the correct parameters set up automatically.

If your web browser does not display the remote form, you should check that the Omnis web client is installed and registered properly.

- Close or minimize the browser when you want to return to Omnis

Wizard Form Wizard

The Wizard form wizard creates a remote form containing a page pane component, and Next, Previous and Finish buttons. The resulting form is rather like a wizard in that it allows the user to enter data on a number of panes using the Next button, and gives you more control over the order in which the user is prompted for information. In the wizard you can add fields and components to each pane in the form. When you have created the form you need to program the Finish button to process the contents of the form.

To create a Wizard Form

- Open the Component Store and click on the Remote Form Classes button
- Select the Wizard remote form wizard and drag its icon onto your open library in the Omnis Class Browser
- Name the new remote form and press Return
- Select an existing remote task from the list of tasks provided, and click on the Next button

or

- Click on the Create New Task option, select a template from the list provided such as the Plain Remote Task, name the new remote task, and click on the Next button
- Enter the number of pages for your form
- Enter the details for each field on the first page, including component type, dataname and text label, clicking on the Add button for each one
- Add the remaining fields for the other pages in your form

The list in the wizard window shows the fields you have added to each page in the form. At any stage, you can click on a field in the list, edit its details, and click on the Update button. You can delete a field in the list by Right/Ctrl-clicking on the field and selecting Delete Field from its context menu.

- When the fields for each page in your wizard-style form are complete, click on the **Next** button
- Select the type of web browser and click on the **Create** button

The wizard now creates the remote form using the details you have entered. Note that Omnis creates a remote task as well.

- To modify the form, double-click on it in the Class Browser

To test the form

- Press Ctrl/Cmnd-T, or Right-click/Ctrl-click on the form and select the **Test Form** option from the context menu

Submit Form Wizard

The Submit form wizard creates a standard submit form containing your fields, Submit and Clear buttons. The wizard lets you add fields and components to the form. In the resulting form, you need to program the Submit button to process the contents of the form.

To create a Submit Form

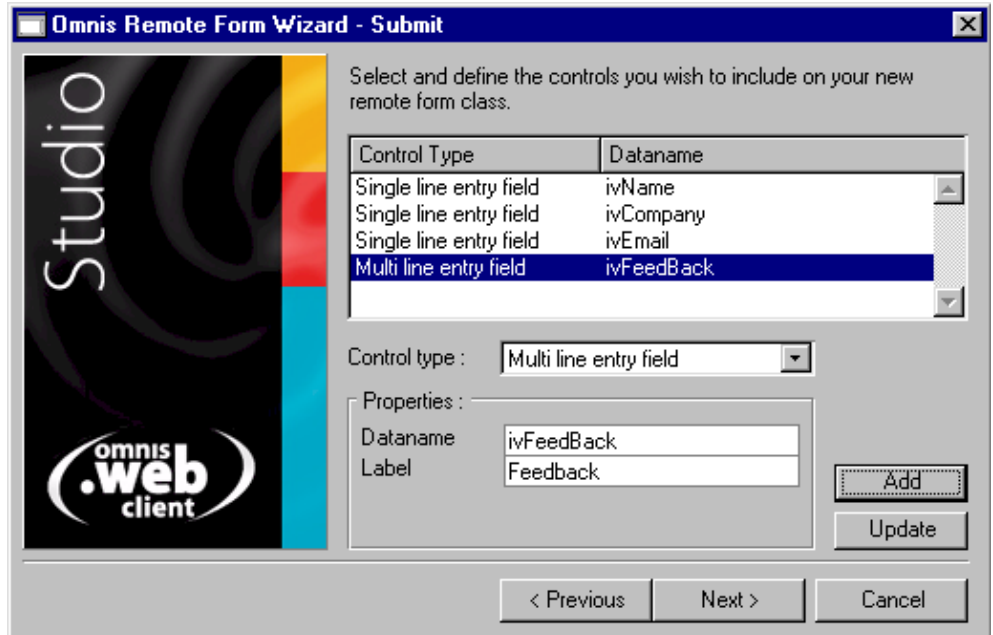
- Open the Component Store and click on the Remote Form Classes button
- Select the Submit remote form wizard and drag its icon onto your open library in the Omnis Class Browser
- Name the new remote form and press Return
- Select an existing remote task from the list of tasks provided, and click on the Next button

or

- Click on the Create New Task option, select a template from the list provided such as the Plain Remote Task, name the new remote task, and click on the Next button
- Enter the details for each field on the form, including component type, dataname and text label
- Use the Add button to add each field in the form

The list in the wizard window shows the fields you have added to the form. At any stage you can click on a field in the list, edit its details, and click on the Update button.

The following screenshot shows a Submit wizard that defines a number of fields for a typical feedback-type form, including single-line entry fields for Name, Company name, E-mail address, and a multi-line field for the Feedback comments.



- When the details for your form are complete, click on the **Next** button
- Select the type of web browser and click on the **Create** button

The wizard now creates the remote form using the details you have entered. Note that Omnis creates a remote task as well.

- To modify the form, double-click on it in the Class Browser

The image shows a window titled "Remote Form TEST.Submit". Inside the window, there are four labels on the left: "Name", "Company", "E-mail", and "Feedback". Each label is followed by an input field. The "Name", "Company", and "E-mail" fields are single-line text boxes. The "Feedback" field is a multi-line text area. At the bottom of the window, there are two buttons: "Submit" and "Clear".

To test the form

- Press Ctrl/Cmnd-T, or Right-click/Ctrl-click on the form and select the **Test Form** option from the context menu

Password Form Wizard

The Password form wizard creates a standard password form containing Password and Username fields. The wizard specifies these two fields by default, but you can change these or add your own fields. In the resulting form you need to program the Submit button to send the contents of the username and password fields to your server application.

To create a Password Form

- Open the Component Store and click on the Remote Form Classes button
- Select the Password remote form wizard and drag its icon onto your open library in the Omnis Class Browser
- Name the new remote form and press Return
- Select an existing remote task from the list of tasks provided, and click on the Next button

or

- Click on the Create New Task option, select a template from the list provided such as the Plain Remote Task, name the new remote task, and click on the Next button

Omnis Remote Form Wizard - Password

Select and define the controls you wish to include on your new remote form class.

Control Type	Dataname
Single line entry field	iPassword
Single line entry field	iUserName

Control type : Single line entry field

Properties :

Dataname	iPassword
Label	Password
Passwordchar	*

Add Update

< Previous Next > Cancel

- If you like, you can change the Password and Username field details using the Update button, or add some more fields as required
- When the details for your form are complete, click on the **Next** button
- Select the type of web browser and click on the **Create** button

The wizard now creates the remote form using the details you have entered. Note that Omnis creates a remote task as well.

- To modify the form, double-click on it in the Class Browser

To test the form

- Press Ctrl/Cmnd-T, or Right-click/Ctrl-click on the form and select the **Test Form** option from the context menu

Sidebar Form Wizard

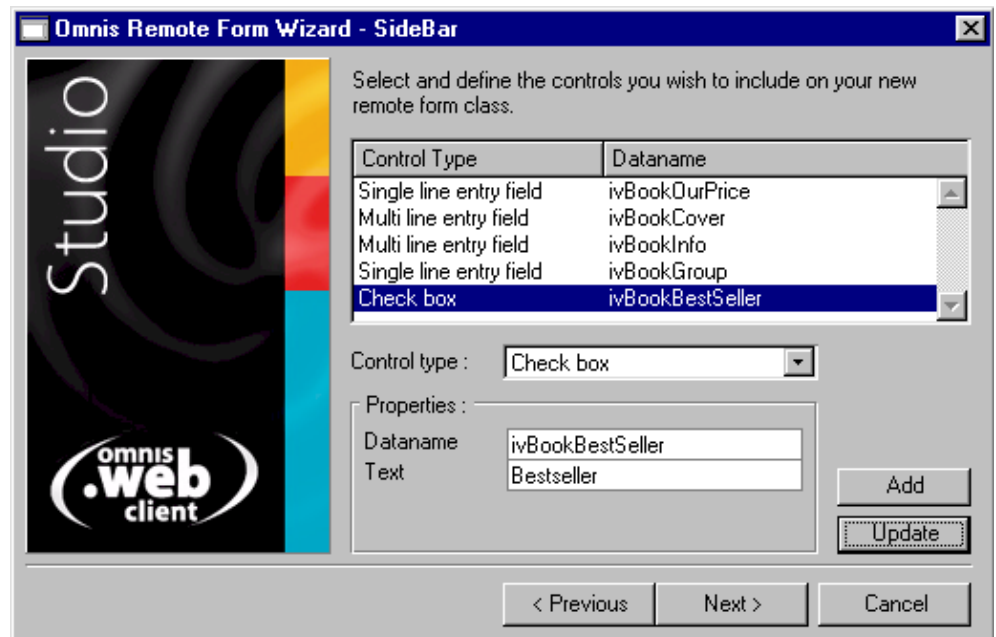
The Sidebar form wizard creates a form containing a number of fields and a sidebar component. In the wizard, you can specify the groups and group items for the Sidebar component. The wizard creates a list containing the group items in your sidebar. When you have created your form, you will have to add some programming behind the sidebar to allow it do respond to clicks or double-clicks.

To create a Sidebar Form

- Open the Component Store and click on the Remote Form Classes button
- Select the Sidebar remote form wizard and drag its icon onto your library in the Omnis Class Browser
- Name the new remote form and press Return
- Select an existing remote task from the list of tasks provided, and click on the Next button

or

- Click on the Create New Task option, select a template from the list provided such as the Plain Remote Task, name the new remote task, and click on the Next button



Omnis Remote Form Wizard - SideBar

Select and define the controls you wish to include on your new remote form class.

Control Type	Dataname
Single line entry field	ivBookOurPrice
Multi line entry field	ivBookCover
Multi line entry field	ivBookInfo
Single line entry field	ivBookGroup
Check box	ivBookBestSeller

Control type :

Properties :

Dataname :

Text :

- Enter the details of the fields you require on the form, and click on the Next button

For the Sidebar form wizard you need to enter the different groups and separate group details. The wizard lets you enter the group name, and the name and icon id for each item in a group. When you click in the IconID field the Select Icon dialog opens which lets you select an icon from the #ICONS system table in the current library, or the Omnispic or Userpic icon data file.

The following screenshot shows three groups and a couple of items in each group.

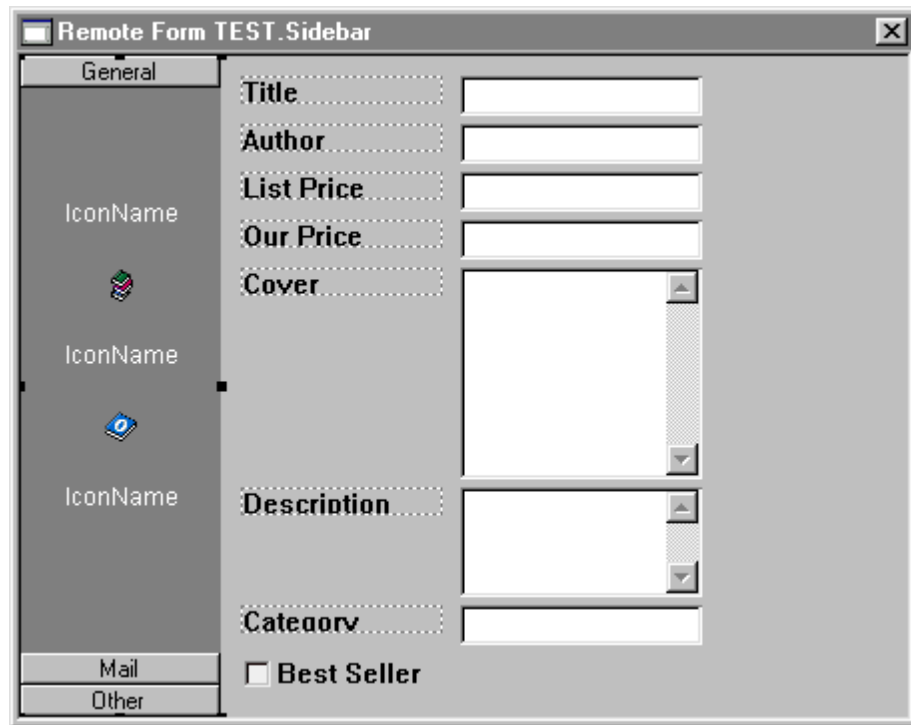


- When your Group details for the form are complete, click on the **Next** button
- Select the type of web browser and click on the **Create** button

The wizard now creates the remote form using the details you have entered. Note that Omnis creates a remote task as well.

- To modify the form, double-click on it in the Class Browser

The following screenshot shows the form created using the details shown above. Note that in design mode your sidebar will not display the group or categories you have just entered in the wizard. The details for the sidebar are retrieved from a list contained in the form and only created at runtime.



To test the form

- Press Ctrl/Cmnd-T, or Right-click/Ctrl-click on the form and select the **Test Form** option from the context menu

Back in design mode, you need to add a method behind the Sidebar component to respond to clicks. For example, you could add the following method to load the list line number of the item clicked and then execute a method using this information.

On evIconPicked

```
Set current list iSideBarList ;; the list behind the sidebar
Load from list {pLinenum} ;; loads linenum of selected item
Do ...something
```


Sidebar with pages Form Wizard

The Sidebar with pages form wizard creates a form containing a page pane and a sidebar component. In the wizard, you can specify the groups and group items for the Sidebar component. The wizard creates a list containing the group items in your sidebar. The wizard lets you specify the fields for each pane and allows you to link each pane to a particular group in the sidebar component.

To create a Sidebar with pages Form

- Open the Component Store and click on the Remote Form Classes button
- Select the **Sidebar with pages** remote form wizard and drag its icon onto your library in the Omnis Class Browser
- Name the new remote form and press Return
- Select an existing remote task from the list of tasks provided, and click on the Next button

or

- Click on the Create New Task option, select a template from the list provided such as the Plain Remote Task, name the new remote task, and click on the Next button
- Specify the number of pages you require on the form, and enter the details for the fields you want on each page, then click on the Next button
- Next you specify the group details for your Sidebar
- When the details for your form are complete, click on the **Next** button
- Select the type of web browser and click on the **Create** button

The wizard now creates the remote form using the details you have entered. Note that Omnis creates a remote task as well.

- To modify the form, double-click on it in the Class Browser

To test the form

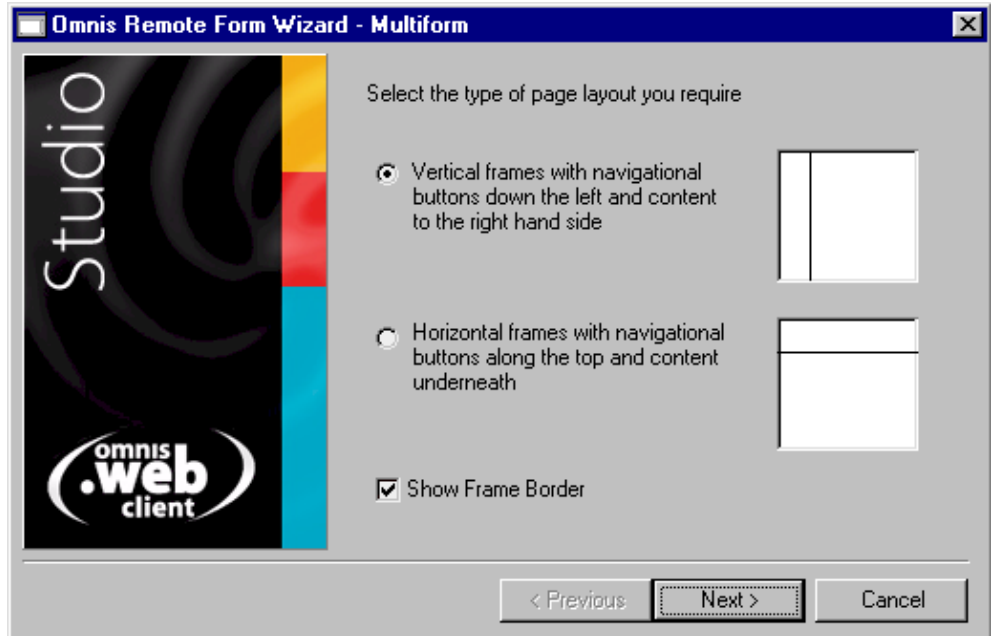
- Press Ctrl/Cmnd-T, or Right-click/Ctrl-click on the form and select the **Test Form** option from the context menu

Multiform Form Wizard

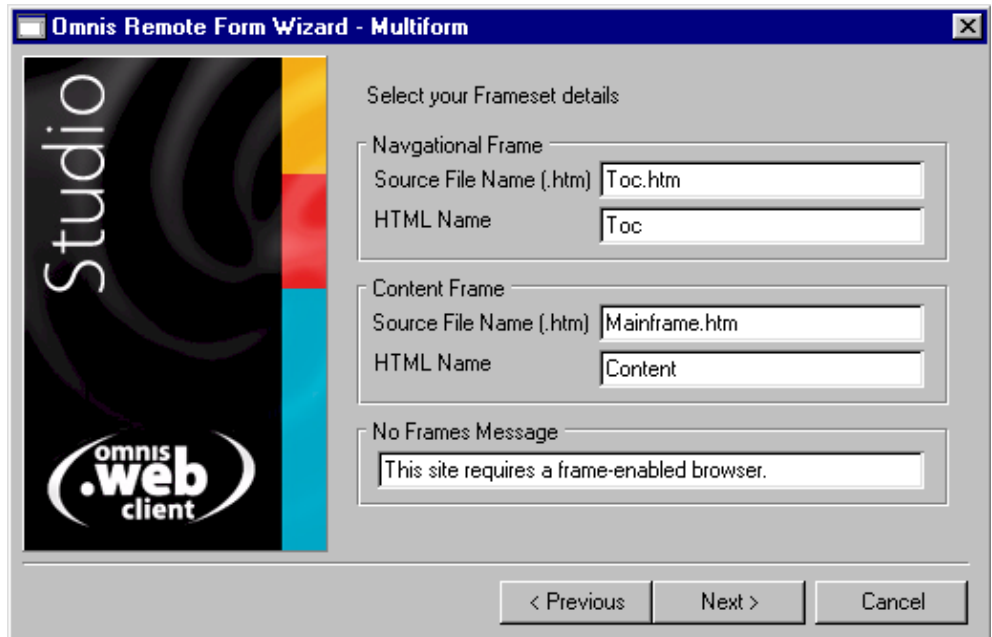
The Multiform wizard lets you create a complete website containing a home page, navigation bar and any number of Omnis remote forms. The wizard creates an html page with a horizontal or vertical frame and navigation buttons linked to the different forms. In the wizard you can select existing remote forms to include in your website, or create new remote forms from the available wizards and templates.

To create a Multi Form website

- Open the Component Store and click on the Remote Form Classes button
- Select the Multiform remote form wizard and drag its icon onto your library in the Omnis Class Browser
- Name the new remote form and press Return



- Select the layout for your html pages, either **Vertical** or **Horizontal** navigation bar, and uncheck the **Show frame border** option if you want to hide the frame border
- When you've selected the layout you require click on the **Next** button

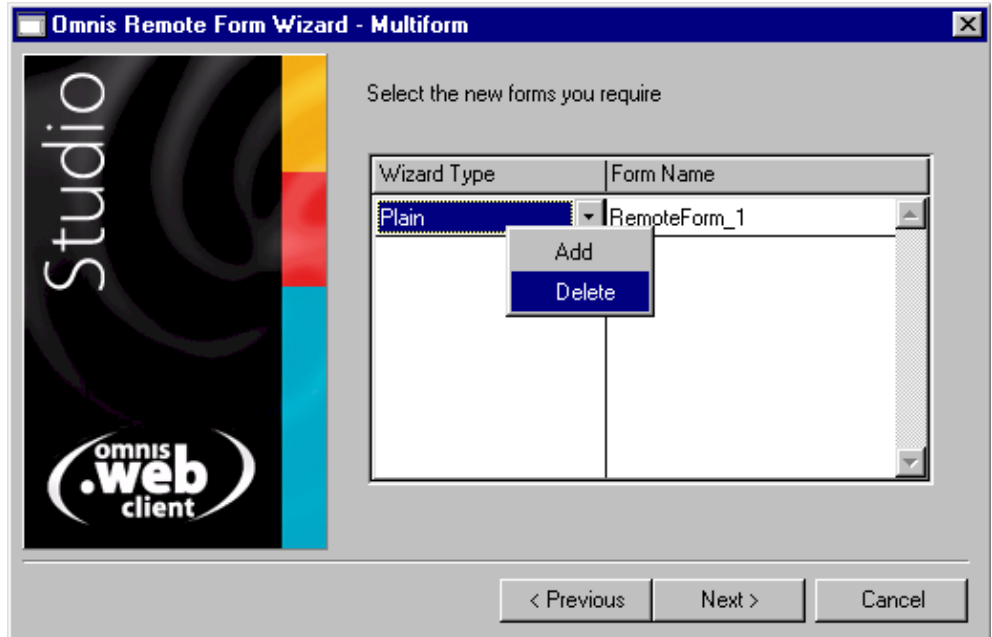


Next the Multiform wizard prompts you to enter the Frameset details. You can enter the name of the html file for the navigation frame or table of contents, and the name of the content frame or Home page. These files are added to the HTML folder in the main Omnis folder, and you can edit them at a later date.

- Enter the file names for the Navigational and Content frames or accept the default name and click on the **Next** button

Next the Multiform wizard lets you select any existing remote forms in your library that you want to include in your website. If your library does not contain any remote forms, click on **Next** to go the next stage straightaway. Otherwise

- Select the name of any existing remote forms in your library that you want to include in your website, and click on the **Next** button



The Multiform wizard now lets you enter a list of new remote forms you may want to add to your website. If your library does not contain any remote forms, you will need to specify a list of new forms in the table, which lets you specify a Wizard type and Form name for each new remote form required.

If you don't want to add any new remote forms you can delete the contents of the list by Right/Ctrl-clicking on the list items and selecting the **Delete** option, as shown above.

- Enter a list of new remote forms you require or clear the list as above, then click on the **Next** button
- Select the type of web browser and click on the **Create** button

If you listed any new remote forms to be created, the appropriate remote form wizards are now launched. When you have completed the wizards for any new remote forms you are returned to the Class Browser. The Multiform wizard creates the specified html pages and adds any new remote forms to your library. To test your multi-form website, you need to open the appropriate html file. To do this:

- Open the HTML folder in your main Omnis folder
- Double-click on the **Frameset.htm** file to open it in your web browser

New Remote Form Template

You can create a remote form from scratch using the New Remote Form template available in the Component Store, or from the **Class>>New** option in the Class Browser. However this type of form does not contain any fields or instance variables, and you have to link it to a remote task manually by specifying its \$designtaskname property. In most cases, the form wizards provide a quicker route for creating remote forms.

To create a new remote form

- Open the Component Store and drag the New Remote Form template onto your library in the Omnis Class Browser
- Name the new remote form and press Return
- Double-click on the remote form in the Class Browser and add all the instance variables, fields, and components as required

Debugging and Testing Remote Forms

In order to run and debug a remote form, the development version of Omnis lets you open a remote form in your local web browser. You can press Ctrl/Cmnd-T while designing your remote form, or pick **Test Form** from the form's context menu, to display your form in your browser. The Omnis web client establishes a connection to the debugging Omnis. You can set breakpoints in the form code and step through the code as usual.

Icons and Icon Pages

Remote form classes have all the standard properties of an Omnis class together with the \$iconpages property, shown under the General tab in the Property Manager.

You can add an icon to several of the remote form components, such as pushbuttons, transbuttons, the tile object, while some of the standard form objects, such as radio buttons, check boxes, and tree lists, also require icons to work correctly. The icons for all the web components in a remote form are sent to the client, together with the class data, when the form is instantiated. However, Omnis does not send individual icons to the client, but complete icon pages as defined in the #ICONS system table in the current library, or the Omnispic or Userpic icon datafile using the Omnis Icon Editor. The \$iconpages property for a form specifies a list of icon pages which are to be sent to the client when the form is opened.

In the Property Manager you can click on the droplist for the **iconpages** property and select a page or a number of icon pages. The names of icon pages are stored in the **iconpages** property separated by commas, and different pages from the #ICONS system table, the

Omnispic, and Userpic icon data files (in that order) are further separated by semi-colons. For example, two pages called Books and General from the #ICONS table in the current library and one page called Embedded Colors from Userpic datafile are stored as in the \$iconpages property as:

Books,General;;Embedded Colors.

You can add icon pages using the notation RemoteFormRef.\$iconpages.\$add(iconID). Omnis adds the name of the page containing the specified icon to the \$iconpages property of the remote form.

Creating your own Icons and Icon Pages

To optimize a remote form, you could create a single icon page containing all the icons needed by the form. In this case, the client only needs to fetch a single icon page, rather than multiple pages. You can create your own icons and icon pages using the Icon Editor and store them in either the #ICONS system table, Omnispic, or Userpic icon datafile. You can copy and paste icons from existing icon pages, but you must be careful what icon id you give new or copied icons. See the *Using Omnis Studio* manual for more details about using the Icon Editor.

Cursors

A useful property of several web components, such as pushbuttons and button areas, is the \$cursor property which lets you assign a cursor icon to the object; when the user's mouse enters the button the cursor changes to the specified icon. You can choose a cursor icon from the Omnispic or Userpic icon data file, or you can add your own icon to the #ICONS system table and assign it to your button. To specify your own icon you need to set \$cursor to kcursCustom and specify the iconid of your icon. Remember that you need to include the name of the page containing your icon in the \$iconpages property of the remote form.

Events

The majority of the web components generate one or more events, although the events for each component are disabled by default. You have to enable specific events for a component by setting its **events** (\$events) property in the Property Manager. When enabled, events are passed to the \$event() method for the component. You can add code to the \$event() method for a component, which is called when the event is triggered.

Event handling methods can be set to run on the client (Right-click on the method and select **Execute on Web Client**), and for most simple methods and calculations this is advisable. By default, an event is sent back to the Omnis server, the client is momentarily suspended while the event handling method is processed on the server, and when the method is finished control is passed back to the client. In many cases, this is not a problem, but for

longer more complex methods this would create an unnecessary delay on the client. It depends what the method has to do and what information it requires: in general, any method that changes the user interface on the client can be executed on the client, while a method that needs to fetch data or write data to your server database needs to execute on the Omnis server. See the *Client Method Execution* section for more information.

When an event is triggered, a number of event parameters are sent from the client to the event handling method. The first of these parameters is always the name of the event that occurred, and all subsequent parameters are specific to the event and describe it in more detail. For example, a click on a list passes the click event in the first parameter (pEventCode=evClick) and the list line clicked on in the second parameter (pLineNumber). For more information about handling events, see the *Omnis Programming* manual.

Standard Field Events

Most web components report the **evBefore** and **evAfter** events, which are triggered when the user is about to enter the field or leave the field, respectively. You can use the *On* command to detect events in your event handling methods. For example, in the \$event() method for a single-line entry field you could use the following commands to detect the evBefore or evAfter event.

```
On evBefore
    ; do something..
On evAfter
    ; do something else..
```

Pushbuttons and Lists

Pushbuttons, button areas, and all the list type components report the **evClick** event, as well as evBefore and evAfter; and some list types also report the **evDoubleClick** event. The Booklist (heading list type) in the tutorial library has the following \$event() method, which is executed when the user double-clicks on a line in the list:

```
On evDoubleClick
    Do method $findnow
        ; executes the $findnow method contained in the form.
        ; the method loads the details for the selected book and
        ; redraws the fields on the form
```

Other web components have their own particular events. For example, the Sidebar component triggers the **evIconPicked**, and **evSetPicked** events; the first one is passed when an icon is selected, the other is reported when the user selects an icon strip or group. The Sidebar component used in the tutorial library has the following \$event() method, which is executed when the user selects an icon:

```

On evIconPicked
  Set current list iSideBarList
    ; makes the list var behind the sidebar the current list
  Load from list {pLinenum}
    ; loads the line from the list using the line number
    ; pLinenum param is sent to the method
  Do method $ctask.$buildbooks (iBookGroup)
    ; executes the $buildbooks method in the current task using
    ; the current value of iBookGroup from the selected list line
  Calculate iBookList as tBookList
    ; transfers the data from tBookList to iBookList instance var
    ; contained in the remote form
  Do $cinst.$objs.BookList.$redraw()
    ; redraws the Booklist field on the form
  Do $cinst.$senddata(iBookList)
    ; sends the iBookList data to the client

```

Remote Form Events

Remote forms generate the `evFormToTop` event only, which you can detect in the `$event()` method for the remote form. See the *Multiple Forms* section later in this manual. Note that in addition, the `$construct()` method of the form is called by default when a form is opened on the client, but in this case no event is generated.

Programming Remote Forms

When writing methods for a remote form class or its components, you must consider that the class may be instantiated many times, depending on how many clients are connected simultaneously. Although you can use any Omnis command or notation and create methods of any length, you must choose carefully what commands you use and how you program certain operations. You can handle events by executing the event handling method on the client machine, or an event may call and execute a method on the Omnis Server. When you design your application, you have to decide which events are best handled on the client or the server, and you need to consider what impact any programming will have on general performance of your application.

You cannot modify a remote form instance at runtime, that is, you cannot add objects to the form instance, or remove them. The same applies to instance variables. These type of changes must be made to the remote form class prior to opening an instance, if they are required. In addition, invalid notation will generate an error on the server and the client will be suspended.

Window related 4GL commands such as 'Redraw' do not work and should not be used in an application using web client. In general, you should try to use notation for the current

object; for example, in the case of redrawing an object, you can use the notation `$cinst.$objs.Object.$redraw()`, `$obj.$redraw()`, or `ObjectName.$redraw()` depending on the context of the current method.

Optimizing Data Handling

After executing an event on the server, by default Omnis returns all instance variables to the client. To optimize what variable data is returned to the client, you can specify which variables are returned using the `$cinst.$senddata(ivVar1,...ivVarN)` method. Once `$senddata()` has been executed, Omnis will only return the data of the specified variables. Note that the `$senddata()` method does not work in a method executed on the client: in all cases, it has to execute on the Omnis Server in order to send the data.

When you change the data for a component, normally it is enough to issue a `$redraw()` for the component using `$obj.$redraw()` to inform the component to reload the data. If you want to optimize data transfer, you will also need to issue a `$senddata()`, specifying the `dataname` of the component. You may execute a `$senddata()` method more than once, and you can specify the same instance variable more than once.

If you issue a `$cinst.$senddata()` without specifying any instance variables, it will clear the send state of previously specified variables, and all variables will be sent, unless you issue another `$senddata()`. If you do issue another `$senddata()`, only the newly specified variables will be sent. The notation `$cinst.$senddata(#NULL)` stops Omnis from sending any data. You can also use `$cinst.$senddata(#NULL,ivVar1)` to ensure that only `ivVar1` is sent.

Note that the next event resets Omnis to the default behavior of sending all data variables in the current remote form.

Open windows and User Prompts

You should not use any commands or programming in your library that would result in a prompt appearing on the Omnis Server. For example, if you use the standard *OK message* command in a server executed method, the message appears on the Omnis Server, not in the client's browser and the server is effectively halted since the message is not cleared until you click OK; see below for method to show a message on the client. If a message or prompt remains open in the Omnis Server all client to server interaction is suspended and the client's browser is locked up. This situation is not serious during development on your local machine, since you can switch to Omnis and clear the message box or prompt, but for an application deployed on the web, this would be disastrous and would severely inhibit useability!

There are many commands that you cannot use for the above reason, they include: *OK message*, *Yes/No message*, *Working message*, *Prompt for input*, *Prompt for data file*, and so on.

You also need to consider the result of using any command or notation that opens a window, installs a menu or toolbar, prints a report, and so on. These commands all do

something in the server application, and the result of any such command is not visible in the client's browser. For example, if you run the *Open window* command, an Omnis window class is opened in the server application and not in the client browser; this is Ok for a server admin window, for example, but is of no use for the client. Therefore, bear in mind that all user interaction over the web must occur using remote forms opened in the client.

OK Messages

You can display a message on the client using the `$showmessage()` method. For example, the following code for a pushbutton opens a message dialog.

```
On evClick    ;; button must have evClick enabled
    Do $cinst.$showmessage('Message','Title')
```

Redrawing Remote forms

The `$redraw()` method allows you to redraw the current remote form instance: `$redraw(bSetcontents=kTrue,bRefresh=kFalse)` redraws all objects on the form. If you pass `bSetContents` as `kTrue`, the data for each object is set using its `$dataname`. Note that this will use the data currently available on the client. If you pass `bRefresh` as `kTrue`, the client draws the form contents, for any objects requiring a screen update. Note that if you pass `kFalse`, the objects requiring a screen update will still be drawn, but not necessarily before `$redraw()` completes.

Opening a Browser window on the Client

The `$showurl()` method allows you to display an Html page in a separate browser window on the client machine: `$showurl(cUrl[,cFrameName,cBrowserProperties])` displays an Html page in a new browser window or frame, where `cURL` specifies the URL of the Html page, and `cFrameName` specifies the Html frame name. If `cFrameName` is empty the page is displayed in a new window, otherwise it is displayed in the specified frame of the current browser window.

The optional parameter `cBrowserProperties` is ignored if `cFrameName` is not empty. Otherwise, it has the same format as the Javascript argument to `window.open`, for example `"toolbar=0,menubar=1"` which specifies that the browser window will have a menubar, but not a toolbar. The possible keywords are:

toolbar	specifies if the browser window has a toolbar
status	specifies if the browser window has a status bar
menubar	specifies if the browser window has a menu bar
scrollbars	specifies if the browser window has scrollbars
resizable	specifies if the browser window is resizable
location	specifies whether the browser window has a location bar
directories	specifies whether the browser window displays Web directories
width	width of browser window
height	height of browser window
top	top coordinate of browser window
left	left coordinate of browser window

The keywords are all Boolean (0 or 1) except for the width, height, top and left, which are numbers in pixel units. Be aware that some browsers do not fully support these properties.

Form Caching

Server Form Caching

In order to improve performance at connection time, the Omnis Server caches various items when the first client of a remote form connects. Subsequent connections will return this cached data. The following items are cached: the class data, the instance variables definition, the icon pages, and the font tables.

You can implement some sort of update process while the system is live, for example, the remote form has been changed, but the cache needs to be cleared. If the cache isn't cleared new connections will continue to use the old class data.

You can use the \$root method \$clearcachedforms() in your application to clear all the cached items on the Omnis Server.

Client Form Caching

In addition to the server form caching, the Omnis web client caches any remote forms it has downloaded from the server on the clients hard disk. The client caches the remote form class data, the instance variable definitions, icon pages, and font tables. The client cache stores the modification dates of each item. When the client connects to the form at a later date, it passes these modification dates to the server, and the server returns only those items that have changed.

IMPORTANT NOTE: The icon pages receive their modified data from the #ICONS system table in the current Omnis library. If you include icons from the OmnisPIC or USERPIC icon datafile and you update them, you must modify the #ICONS class to force all icons on the client to be updated; there is no other way for the Omnis Server to tell if icons in Omnispic.df1 or Userpic.df1 have changed, other than checking the modification date of #ICONS.

Client Method Execution

You can specify that a remote form method is to run on the client rather than the server. In particular, you can perform simple calculations or process events on the client, thus avoiding the overhead of sending events and messages between the client and server, reducing the network traffic and workload on the server. Methods that run on the client can contain only a subset of the complete Omnis command language. Command groups that allow client execution include:

- Calculations...
- Constructs...
- Methods...
- Events...
- Message boxes...

To set a command to client execution, Right-click/Cmnd-click on the remote form's method name and select the **Execute On Web Client** popup menu option. When you try to specify that an existing method should execute on the client, Omnis checks to see if it contains any commands that cannot be run on the client. In this case, Omnis returns an error and names the offending command, or the first command if the method contains many commands that do not allow client execution. If the method can be run on the client it is shown in pink (the default) and the command list (bottom left of method editor) is modified to include only those commands that can be executed on the client.

To optimize your web client application, you should perform as much processing as you can on the client, and only when necessary pass method execution and instance variable values back to the server. You should avoid making calls to the server from within client methods, and vice versa, to reduce the network traffic and possible delays in the client. It works both ways: when you design your methods you should try to contain all client-side scripting on the client and all server-side processing on the server and not constantly pass method calls, data, or events back and forth during a single transaction or event processing action. For example, if you use *Do method* command on the client, only call other methods that can be executed on the client, avoid calling a method that executes on the server unless absolutely necessary. In addition, some remote form methods, such as \$senddata(), will not work if they are contained in a method that executes on the client.

Methods have the property \$clientexec which is kTrue if the method is set to run of the client, or kFalse if it should run on the server.

Multiple forms

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

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

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

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

There are some further restrictions to note:

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

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

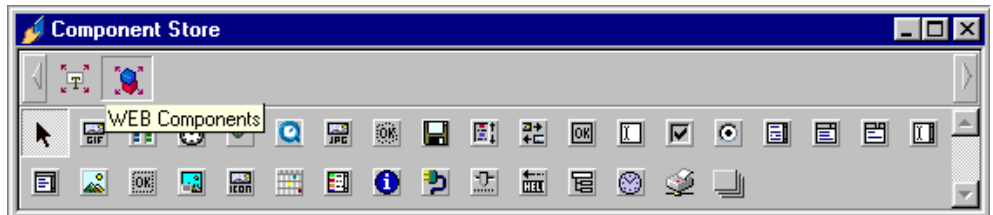
To facilitate communication between different remote form instances, remote forms can also receive one event, `evFormToTop`. In design mode, you can enable this event for a form, using the `$events` property of the form. The event generates a call to `$event()` method of the remote form. `evFormToTop`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

Chapter 4—Remote Form Components

The Omnis Component Store contains over 30 different web components that let you create very interactive, feature-rich environments for your web applications. The remote form wizards add components to your forms automatically, but if you are creating your own forms or adding fields to an existing form, the Omnis Component Store provides many types of component. When you have created a component in your remote form, you can modify it by changing its properties in the Property Manager.

To create a web component from the Component Store

- Open your remote form in design mode
- Press F3/Cmnd-3 to open the Component Store or to bring it to the top; note the Label, Border, Wash, and Tile objects are in the WEB Background Objects group which is displayed by default
- Click on the WEB Components button in the Component Store



- Drag the required component type from the Component Store and drop it onto your remote form

or, to draw a web component of a particular size

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

or, to place a component in your form automatically

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

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

All the web components are located in Studio under the WEB Components button in the Component Store, and are visible only when you open a remote form in design mode. The background components such as the label, border, wash, and tile objects are located under the WEB Background Objects button in the Component Store.

For development purposes, all the web components are installed in the Webcomp folder in the Omnis tree; any new components should be added to this folder. If you add more components to your Webcomp folder you have to register them via the Component Store by right-clicking of the Store and selecting the External Components option. From the user's point of view, the default Omnis web client installer contains the standard field components only, including the entry field, check box, radio button, single-column list, and so on. All the other components are downloaded to the user's browser when they are required by a particular remote form.

The majority of the web components have their own Custom tab in the Property Manager containing properties specific to the component. Where a component requires a \$dataname, such as an entry field or list, you must assign an instance variable to the remote form field; you cannot use a file class field directly in a remote form field. You can however assign a file class field value to an instance variable and display this in your remote form. For example, you can load a list from file and assign it to an instance variable of list type and use the list instance variable to populate a sidebar component.

For further details about creating fields and components using the Component Store and modifying their properties via the Property Manager, see the *Using Omnis Studio* manual.

The following sections introduce the different components you can use in remote forms. Each section provides a few code samples showing how you can program the component. Much of the sample code is taken from the example libraries that are available in the Web Component Gallery on www.omnis.net.

Web Component Properties

In general, the properties of a web component are unique to the object and their names will not clash with standard Omnis field properties. However, when a web component property has the same name as a standard built-in Omnis field, you must access the external property using a double colon (::) before its name. For example, the tab bar component has the property \$currenttab which you must refer to using the notation:

```
Do $cinst.$objs.tabbar.$::currenttab.$assign(2)
```

Border

The Border component is contained in the FORMBACK component file, and is installed along with the basic field components in the Omnis web client installer.

The border component lets you create a rectangle or square with an inner and outer border edge style. You can also set the background color. Note the Border object has no methods or events.

Calendar

The Calendar component is contained in the FORMCAL component file. It can present the current month and today's date in a standard calendar format. The component has many useful properties that let you control the overall appearance of the calendar, such as fonts, colors, and the display style for days and headers. To display today's date in runtime, you need to set the \$currday property. For example, to set the current day of the calendar component to today's date you could use:

```
; method contains instance var iCurrentDate (DateTime) with default
value of #D
Do $cinst.$objs.calendar.$currday.$assign(iCurrentDate)
```

The following example remote form contains the calendar component, lists to set the month and year, and buttons to display the next or previous months.

Sun	Mon	Tue	Wed	Thu	Fri	Sat
26	27	28	29	30	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31	1	2	3	4	5	6

The following methods initialize the calendar form and format the display of the calendar component.

```
; $construct() method of the calendar remote form
; form contains iCurrentDate (DateTime), iMonth (Char), iMonthList
(List), iYear (Number), iYearlist (List)
Calculate iCurrentDate as #D
Do method $setdroplists
```



```

; $setdroplists method in the form
Do iMonthList.$define(iMonth)
Calculate assignloop as dpart(kMonth,iCurrentDate)
For loop from 1 to 12 step 1
    Calculate tmp as con("1 ",loop," 90")
    Do iMonthList.$add(dname(kMonth,tmp)) Returns line
    If assignloop=loop
        Do iMonthList.$line.$assign(iMonthList.$linecount)
        Calculate tmp as dadd(kMonth,-1,tmp)
        Do $cinst.$objs.back.$text.$assign(
            con("< ",dname(kMonth,tmp)))    ;; sets the Previous butt
        Calculate tmp as dadd(kMonth,2,tmp)
        Do $cinst.$objs.next.$text.$assign(
            con(dname(kMonth,tmp)," >"))    ;; sets the Next butt
    End If
End For
Do iYearList.$define(iYear)
Calculate assignloop as dpart(kYear,iCurrentDate)
For loop from 1970 to 2030 step 1
    Do iYearList.$add(loop) Returns line
    If assignloop=loop
        Do iYearList.$line.$assign(iYearList.$linecount)
    End If
End For
Do $cinst.$objs.calendar.$currday.$assign(iCurrentDate)

```

The calendar example form shown above has a list for selecting a month and one for the year. The methods for these two lists is as follows:

```

; $sevent() method for the Month selection list
On evClick
    Do iMonthList.$line.$assign(pLineNumber)
    Do iMonthList.$loadcols()
    Calculate iCurrentDate as con(dpart(kDay,iCurrentDate),
        ' ',iMonth,dpart(kYear,iCurrentDate))
    Do method $setdroplists
    Do $cinst.$objs.month.$redraw()    ;; redraw the lists
    Do $cinst.$objs.year.$redraw()

```

```

; $event() method for the Year selection list
On evClick
    Do iYearList.$line.$assign(pLineNumber)
    Do iYearList.$loadcols()
    Calculate iCurrentDate as con(dpart(kDay,iCurrentDate),
        ' ',dpart(kMonth,iCurrentDate),' ',iYear)
    Do method $setdroplists
    Do $cinst.$objs.month.$redraw()    ;; redraw the lists
    Do $cinst.$objs.year.$redraw()

```

In addition, the calendar example form has pushbuttons for showing the Next and Previous months. The \$event() method for the Previous button is as follows:

```

On evClick
    Calculate iCurrentDate as dadd(kMonth,-1,iCurrentDate)
    ; or Calculate iCurrentDate as dadd(kMonth,1,iCurrentDate)
    ; to advance the month by one
    Do method $setdroplists
    Do $cinst.$objs.month.$redraw()    ;; redraw the lists
    Do $cinst.$objs.year.$redraw()

```

The calendar component reports two events: evDateChange, sent when the date changes, and evDateDClick, sent when the user double-clicks on a date cell. Both these events pass the pCurrentDate event parameter. You can detect these events in the \$event() method of the component; assuming you're using a variable iCurrentDate as above you could do:

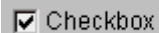
```

On evDateChange
    Calculate iCurrentDate as pCurrentDate
    ; then Do something else

```

Check box

The Check box component is contained in the FORMFLDS component file, and is installed on the client by the default Omnis web client installer.



Check boxes represent boolean data, that is, they can display On or Off choices, Yes or No, and 1 or 0 values. The instance variable you associate with a check box should be of Number or Boolean type and should be assigned in the \$dataname property. You enter the text to display to the right of the check box in the \$text property for the component. Checking a check box in the client sets the value of a numeric variable to 1, or for a boolean variable to True; unchecking it sets a numeric value of 0 (zero) or kFalse.

Note that the images for the check box on and off states (i.e. the box ticked and not ticked) are stored in the Omnispic icon data file in the 'Multistate 2' page (Multistate 3 for MacOS)

and for these to display on the client you have to add this icon page to the \$iconpages property for your remote form.

Check boxes report the evClick event, but no parameters are passed. You can detect the evClick event in the \$event() method for the object.

For example, the login screen in the Banking web client example lets the user enter a username and password, and the user has the option to store their details by checking a check box. The check box is associated with the iRemember which is a Boolean variable. When the user checks the box, the variable is set to true and this value is used in the methods that set up the user account; specifically, if the check box is set, the user information is stored on the client machine using the Info (cookie) component. Here is the method that handles this:

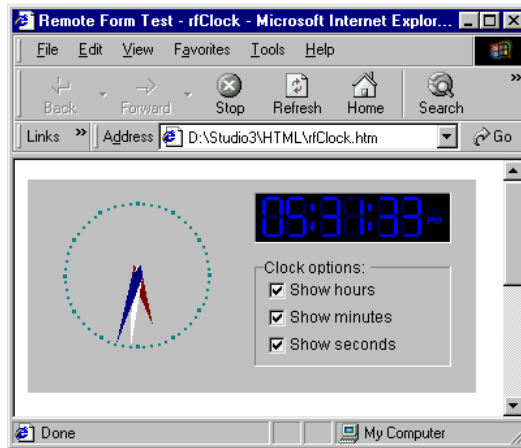
```
; part of the $checkuser method
If iRemember      ;; if check box checked then store user info
    Calculate $cinst.$objs.cookie.$userdata
        as con(iUserName,"@:@",iPassword)
End If
```

Clock

The Clock component is contained in the FORMCLOCK component file and lets you show the current time in the current time zone on the client.

The clock component has many properties, most of them self-explanatory, that let you control the appearance. For example, you can specify whether the clock face is analog or digital using \$digital and display it in 24 hour mode using \$24hour, you can change the colors of the hour, minute, and second hands and the background color, you can set the time zone to be displayed using \$timezone and \$timezoneadj, and you can add an icon to the face using \$iconface, \$iconid, and \$scaleicon.

The clock component passes 3 events detected in the \$event() method for the object: evHoursChange, evMinutesChange, and evSecondChange which in turn pass the parameters pHours, pMinutes, and pSeconds respectively.



The example remote form above has an analog and digital clock face and lets you show or hide the hours, minutes, and seconds. The \$construct() method of the form is as follows:

```
; form contains instance vars iShowHours,iShowMins,iShowSecs (all
  Boolean type)
Calculate iShowHours as kTrue      ;; set the default settings
Calculate iShowMins as kTrue
Calculate iShowSecs as kFalse
Do method $setprops

; $setprops method sets the 2 clocks according to
; current check box settings
Calculate $cinst.$objs.clock.$showhours as iShowHours
Calculate $cinst.$objs.clock.$showminutes as iShowMins
Calculate $cinst.$objs.clock.$showseconds as iShowSecs
Calculate $cinst.$objs.digiclock.$showhours as iShowHours
Calculate $cinst.$objs.digiclock.$showminutes as iShowMins
Calculate $cinst.$objs.digiclock.$showseconds as iShowSecs
```

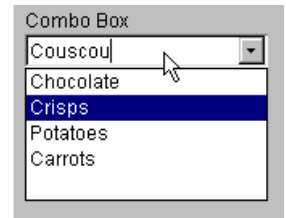
Each of the check box options on the remote form has the following \$event() method:

```
On evClick
  Do method $setprops
```

Combo box

The Combo box component is contained in the FORMFLDS component file, and is installed on the client by the default Omnis web client installer.

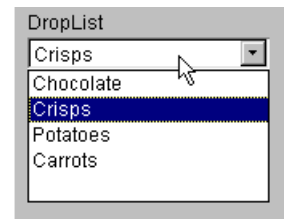
A combo box is a combination of droplist and an entry field. When displayed in the form, the user can choose an item from the list or type anything they want into the entry box. When you create a combo box, you must specify the name of the data entry field in the \$dataname property and the name of your list instance variable in the \$::listname property. For lists variables that contain multiple columns, the \$listcolumn property lets you specify the column in the list that is used to populate the droplist. Combo boxes report the evClick event which you can detect in the \$event() method for the object. This event passes the parameter pLineNumber containing the line number of the list row clicked.



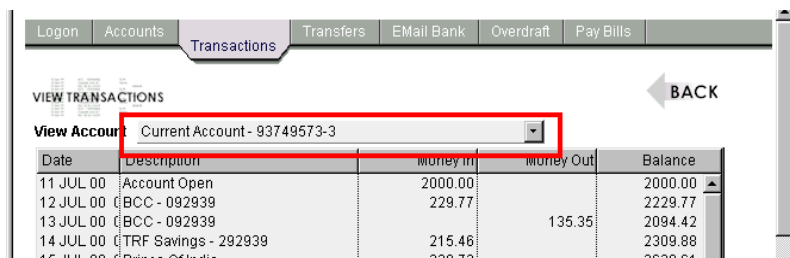
Droplist

The Droplist component is contained in the FORMFLDS component file, and is installed on the client by the default Omnis web client installer.

Droplists let you display list data and allow the user to make a selection; the current selection is displayed in the field. The contents of the list is supplied from an instance variable specified in the \$dataname property. For lists variables that contain multiple columns, the \$listcolumn property lets you specify the column in the list that is used to populate the droplist. Droplists report the evClick event which you can detect in the \$event() method for the object. This event passes the parameter pLineNumber containing the line number of the list row clicked.



The following example method is from the droplist that lets you select your account on the Transaction pane of the Banking web client example; the droplist is highlighted in red.



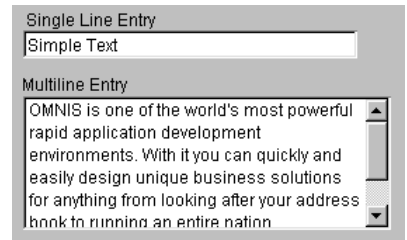
The method behind the droplist loads the Account Id according to the selected line, assigns the stored values to the variables in the form instance, and redraws the fields on the client.

```
; $event() method for droplist
; iAccountList2 is the inst var for the droplist
On evClick
    Single file find on fAccounts.id (Exact match)
        {iAccountList2.[pLineNumber].2}
    Calculate iAccountName as fAccounts.type
    Calculate iTransactions as fAccounts.transactions
    Calculate iFinalBalance as fAccounts.balance
    Do $cinst.$senddata(iTransactions,iAccountName,iFinalBalance)
    Do $cinst.$objs.transaction.$redraw()
    Do $cinst.$objs.accounts.$redraw()
    Do $cinst.$objs.view_accname.$redraw()
    Do $cinst.$objs.final_balance.$redraw()
```

Edit fields

The single-line and multi-line edit field components are contained in the FORMFLDS component file, and are installed on the client in the default Omnis web client installer.

The basic package of components provides two types of field component: the single line entry field and the multi-line entry field. They behave in a similar way and have many properties in common, except that multi-line fields can have scroll bars. The data container for an entry field must be an instance variable specified in the \$dataname property of the field.



Entry fields can display text from your database and they also allow user input. You can set the \$enabled property of a field to kFalse to make it display only. Another useful property is \$passwordchar which lets you specify a single character, such as # or *, to make the entry field into a password field by visually hiding whatever is entered on the client.

Single-line and multi-line edit fields have the \$fieldstyle property which is set to OMNISweb by default. This ensures that all your fields have the same font settings, and that when your form is viewed on different platforms the fonts are mapped to an appropriate font for the current OS. If you want to change the font for a particular field, you need to set the \$fieldstyle to None and assign your own font and style properties. You can set the color of the text in \$textcolor.

Single-line edit fields have some additional properties: `$displayformat` lets you format the data to display a date, or date-time data; and `$autotablen` specifies the number of characters a field is allowed before Omnis tabs automatically.

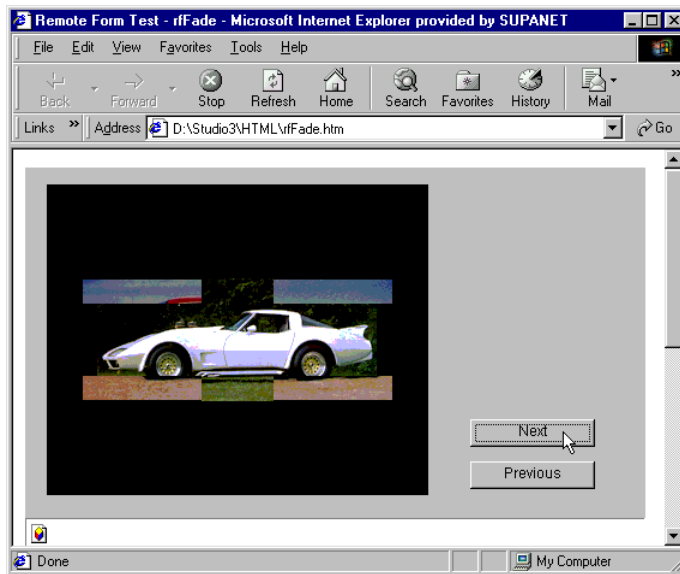
Fade

The Fade component is contained in the FORMFADE component file. The Fade object displays image data and lets you apply a fade effect to an image, using one of over 30 different styles including slide up, slide down, wash left, wash right, spiral in, spiral out, and so on. For example, you could apply a fade effect from one image to the next, as you step through an image database.

The Fade component can display picture data retrieved from either a server database or Omnis data file; the latter supports the new true color shared picture mode. The component requires an instance variable of Picture type specified in the `$dataname` property. The style of fade is stored in the `$fadestyle` property which you can set in design mode, under the Custom tab in the Property Manager, or at runtime using the notation. If set to `kTrue`, the `$fadeondatachange` property ensures a fade is triggered when the image data changes in the object. The `$dissolvesize` property affects the size of the blocks in the fade when the fade style is one of the dissolve styles. Images are displayed same size unless `$stretch` is set to `kTrue` and in this case they are stretched to fit the field size; `$borderh` and `$borderv` let you add a border inside the field when `$stretch` is `kTrue`.

As well as the `evBefore` and `evAfter` field events, the fade component reports the `evFadeFinished` event which you can detect in the `$event()` method for the object.

The following example shows a simple picture database, but rather than using the regular picture field the images are displayed using the Fade component; when the image changes the new image is ‘faded’ using one of the fade styles.



The `$construct()` method in the remote form opens the picture database and loads the first record; the image is stored in the `carPict` field in the `fCars` file class. Note that the images are stored in the library, which is loaded using `sys(10)`. The method then assigns the data to the `iPicture` variable and the variable data is sent to the client.

```
; $construct() method of the form
Open data file (Do not close other data) {[sys(10)],cardata}
Set main file {fCars}
Find first
Calculate iPicture as fCars.carPict
Do $cinst.$senddata(iPicture)
```


The remote form contains a Next and Previous so you can cycle through the database; if you reach the beginning or end of the database the last or first image is loaded to give the appearance of a continuous stream of images.

```
; $event() method for the Next button
On evClick
    Next                ;; or Previous
    If flag false
        Find first      ;; or Find Last for Previous button
    End If
    Calculate iPicture as fCars.carPict
    Do $cinst.$senddata(iPicture)
    Do $cinst.$objs.formfade.$redraw()
```

As an extra refinement you could add two lines of code to the Next and Previous methods to fade the image using one of the 35 fade styles picked at random using the `randintrng()` function. Note that you can set the `$fadestyle` using one of the style constants, such as `kFadeBounce`, or using their numeric equivalent; the group of fade style constant are numbered 1 to 35 in the order they are listed in the Property Manager. The method for the Previous button would now be:

```
; $event() for the Previous button
; form contains iFadeStyle of Number type
On evClick
    Previous
    If flag false
        Find last
    End If
    Calculate iFadeStyle as randintrng(1,35)
    Do $cinst.$objs.formfade.$fadestyle.$assign(iFadeStyle)
    Calculate iPicture as fCars.carPict
    Do $cinst.$senddata(iPicture)
    Do $cinst.$objs.formfade.$redraw()
```

File read/write

The File read/write component is contained in the FORMFILE component file, and provides a pushbutton that lets the user read or write files from the client machine, i.e. download or upload files. The component can be activated by the user clicking on the button, or programmatically by assigning to the `$action` property. The `$dataname` property specifies the name of the instance variable which stores the contents of the file to be read or written. The `$buttonstyle` property defines the appearance of button, while the `$iconid` specifies its icon; note you need to include the name of the page containing the icon in the `$iconpages` property of the remote form. The `$filename` property specifies the default file

name displayed when the user is prompted to save a file to disk, and \$title specifies the title of the navigation dialog.

The \$typelist property of the File component is a comma separated list of file types specifying the description and extensions (Windows) or file types (Macintosh) and displayed in the navigation dialog when the user is prompted to select a file. The type list is also used when the user is prompted to save a file to disk. The list has the following format:

Description, Windows Extension, Macintosh file type,...

For example:

"JPEG Files,.jpg,JPEG,Portable Network Graphics,.png,PING"

The \$filecreator property (MacOS only) specifies the file creator for the file to be created when the user saves a file to disk.

Reading files

To read files in response to user clicks, the \$action property of the File component must be set to kFFRead. When the user clicks the button, the user is presented with a standard navigation dialog, to browse their machine. When the user has picked a file, the component reads the contents of the file into the instance variable specified in \$dataname, and sends an event to the Omnis Server, if the event has been enabled. If the event is disabled, the data is sent to the server the next time another event is sent.

To read files programmatically, the \$action property of the File component must be set to kFFReadNow. The user is prompted to select a file via a standard navigation dialog on the client machine. Once the user has selected a file, the component reads the contents of the file into the instance variable specified in \$dataname, and sends an event to the Omnis Server, if the event has been enabled. If the event is disabled, the data is sent to the server the next time another event is sent.

Note that under MacOS you can use kFFReadEntireFile or kFFReadEntireFileNow to include the data fork and resource fork when the file is read.

Writing files

To write files in response to user clicks, the \$action property of the File component must be set to kFFWrite. The instance variable specified in \$dataname must already contain the file's data, prior to the user clicking on the button to create a file. To write a file programmatically, the instance variable specified in \$dataname must already contain the file's data, prior to the \$action property being assigned the kFFWriteNow constant.

Note that under MacOS you can use kFFWriteEntireFile or kFFWriteEntireFileNow to include the data fork and resource fork when the file is written. The given data to be written must already be in the correct format, i.e. it must contain both the resource part and data part of the file.

Events

The File component reports the standard button field events (evBefore, evAfter, evClick, and evDoubleClick) as well as the following. Note you have to enable these events in the \$events property of the File component.

☐ **evFileRead**

sent to the server when the user has successfully selected a file and the file's contents has been read. pFileName specifies the full path and file name of the file which was read. pFileType and pFileCreator (both MacOS only) the file type and creator.

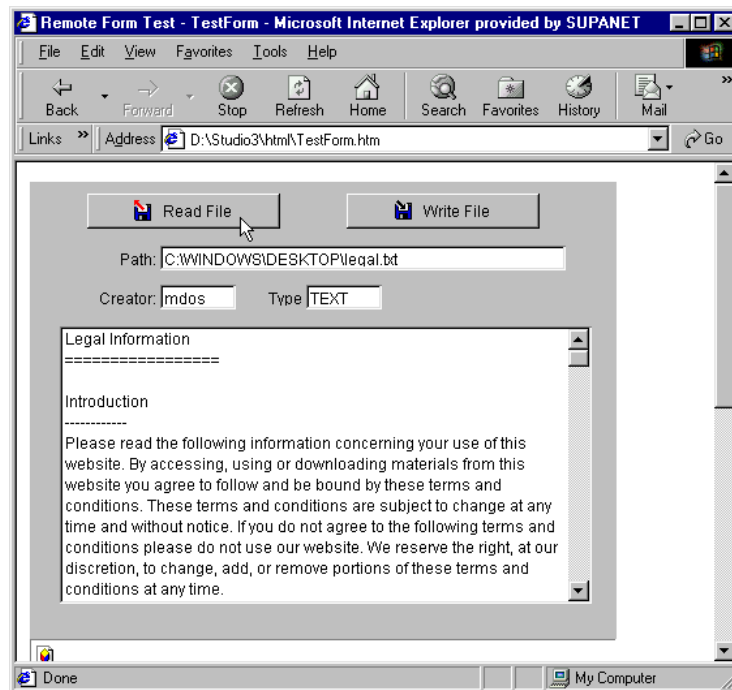
☐ **evFileWrite**

sent to the server when the user has successfully saved a file and the files contents have been written. pFileName specifies the full path and file name of the file which was written. pFileType and pFileCreator (both MacOS only) the file type and creator.

☐ **evFileError**

sent to the server if an error occurred during file reading or writing. pFileErrorCode specifies the error code which are the same as for the FileOps external component.

The following example remote form allows the user to load files from disk and view and edit them on screen. The 'Read File' button is a File component that prompts the user to locate a file.



Rather than loading the selected file in a variable the data, file name, creator, and file type are loaded into fields in the form. The method behind the Read File button is as follows.

```
On evFileRead      ;; a file has been read on the server
    ; display the filename, creator and type
    Calculate ivFileName as pFileName
    Calculate ivFileCreator as pFileCreator
    Calculate ivFileType as pFileType
    Calculate ivFileData as pFileData
    Do $cinst.$senddata(ivFileName,
        ivFileCreator,ivFileType,ivFileData)
    Do $cinst.$redraw()
On evFileError      ;; if an error has occurred
    Do $cinst.$showmessage(con('Error ',
        pFileErrorCode,' when reading file'))
```

As an alternative, you can use a standard button and prompt the user to locate a file by assigning to the \$action property of a File component. The following method can be used behind a standard pushbutton; note in this case you should use the kFFReadNow action.

```
On evClick
    Do $cinst.$objs.ReadFile.$action.$assign(kFFReadNow)
```

Gif

The Gif component is contained in the FORMGIF component file, and can be used to display static or animated GIFs. You can include gif images in your Html pages, but the Gif component lets you display gif images inside your remote forms, or manage a gif image database.

You can specify the gif to be displayed via the \$path property, or the component can get its data from an instance variable specified in \$dataname. You can store or display gif images in a database using the Picture data type, or you can use the Binary type for animated gifs.

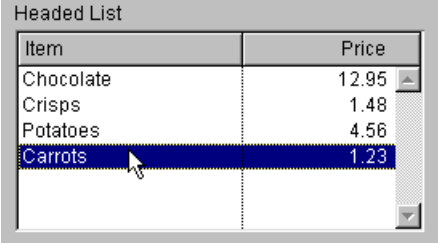
The Gif component reports no special events and has no methods, apart from those for a standard field. You can change the behavior of the field by changing its properties under the Custom tab in the Property Manager. In particular \$picturealign lets you position images inside the field border, for example, you can use kPALcenter which may be useful if your images are all shapes and sizes. In addition, you can enable the \$usepalette property to force Omnis to use the color palette stored with the gif image.

Headed list

The Headed list component is contained in the FORMFLDS component file, and is installed on the client by the default Omnis web client installer.

Headed lists present list data in a multi-column, tabular format with button style column headers. The contents of the list is supplied from an instance variable specified in the \$dataname property. Headed lists have several useful properties under the Custom tab in the Property Manager. The \$aligncolumns property lets you specify the alignment for text in the columns of the list; you can assign a string in the form LLRR where L is for left alignment and R is for right alignment, using one letter for each column. The \$alignheadings property lets you align the column headings and is specified in the same way. In addition, headed lists have the \$displayformat property that lets you format the data to display a date, or date-time data.

Headed lists report the evClick and evHeaderClick events which you can detect in the \$event() method for the object. The evClick event passes the parameter pLineNumber containing the line number of the list row clicked. The evHeaderClick passes the parameter pColumnNumber containing the number of the column clicked.



Item	Price
Chocolate	12.95
Crisps	1.48
Potatoes	4.56
Carrots	1.23

The following remote form is from the Banking web client example and uses a headed list to show the accounts for the current user.

Remote Form Test - rForm - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Refresh Home Search Favorites History Mail Print Edit

Links Address C:\Omnis24\HTML\rfForm.htm Go

Logon **Accounts** Transactions Transfers Email Bank Overdraft Pay Bills

PLEASE CHOOSE AN OPTION

VIEW TRANSACTIONS TRANSFER MONEY EMAIL BANK CHANGE OVERDRAFT PAY BILLS

Current User Name Asmith Stop Auto Logon

Current Available Accounts

Account	Type	Overdraft	Balance
93749573-4	Current Account	900.00	-1309.99
89374628-4	Savings Account	0.00	1632.45
66538462-4	High Rate Account	0.00	2313.94

Double click on an account to see transactions

The list that stores the account information is called `iAccountList` and is built when the form is opened after the user logs on. The list contains the Account number or Id, the account type i.e. Current / Savings / High Rate, the overdraft, and the current balance. The following method builds the account data, and assigns it to the headed list in the form.

```

; $buildaccountlist() method
Set current list fUsers.accounts
; Fixed list of available account types
Do iAccountList.$define(
    fAccounts.id,fAccounts.type,
    fAccounts.overdraft,fAccounts.balance)
; define any other lists required in the form
For each line in list from 1 to #LN step 1
    Load from list
    Single file find on fAccounts.id (Exact match)
        {fUsers.accounts.[#L].1}
    Do iAccountList.$add(
        fAccounts.id,fAccounts.type,
        fAccounts.overdraft,fAccounts.balance)
    ; build any other lists required in the form
End For
Do $cinst.$senddata(iAccountList)
Do $cinst.$objs.accounts.$redraw() ;; the account list in the form
; redraw any other fields as required

```

The headed list in the banking example allows the user to double-click to view the transactions for the selected account. The \$event() method for the headed list detects the evDoubleClick event, builds the transaction, and switches panes in the form.

```

On evDoubleClick
    Do $cinst.$senddata(#NULL)
    Single file find on fAccounts.id (Exact match)
        {iAccountList.[pLineNumber].1}
    Calculate iTransactions as fAccounts.transactions
    Calculate iFinalBalance as fAccounts.balance
    Do $cinst.$objs.pagepane.$currentpage.$assign(4)
    Do $cinst.$objs.tabbar.$::currenttab.$assign(3)
    Do $cinst.$objs.transaction.$redraw()
    Do $cinst.$objs.view_account.$redraw()
    Do $cinst.$objs.final_balance.$redraw()
    Calculate tmp as iAccountList.[iAccountList.$line].1
    Do iAccountList1.$search(fAccounts.id=tmp) Returns found
    Do iAccountList1.$line.$assign(found)
    Do $cinst.$senddata(iTransactions,iAccountList1,iFinalBalance)

```

The transactions for the selected account is also displayed in a headed list on the fourth pane of the page pane in the remote form.

Remote Form Test - rForm - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Refresh Home Search Favorites History Mail Print Edit

Links Address D:\Omnis24\HTML\rfForm.htm Go

Logon Accounts **Transactions** Transfers EMail Bank Overdraft Pay Bills

VIEW TRANSACTIONS

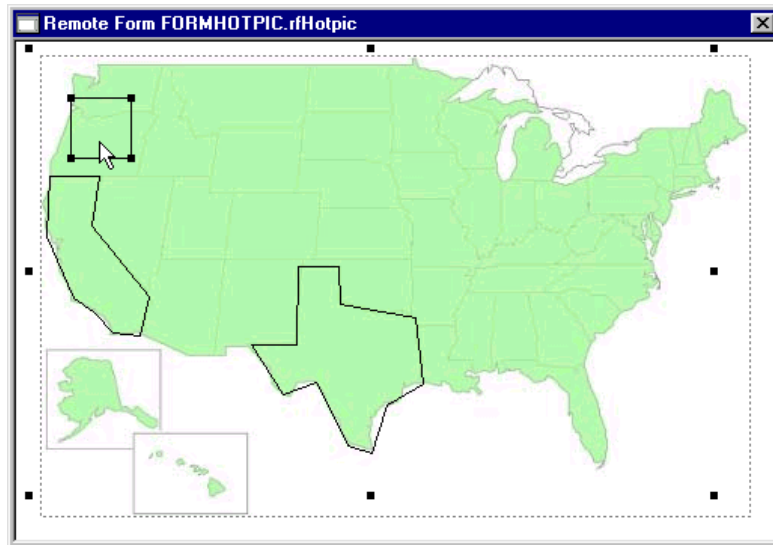
View Account Current Account - 93749573-3

Date	Description	Money In	Money Out	Balance
11 JUL 00	Account Open	2000.00		2000.00
12 JUL 00	(BCC - 092939	229.77		2229.77
13 JUL 00	(BCC - 092939		135.35	2094.42
14 JUL 00	(TRF Savings - 292939	215.46		2309.88
15 JUL 00	(Prince Of India	328.73		2638.61
16 JUL 00	(Carpet City REF:39484		117.94	2520.67
17 JUL 00	(TRF Savings - 292939	222.86		2743.53
18 JUL 00	(Tesco REF: 309349494		184.59	2558.94
19 JUL 00	(DDB - 27273992	197.26		2756.20
20 JUL 00	(ESSO : REF 38938494		270.76	2485.44
21 JUL 00	(Tesco REF: 309349494	19.97		2505.41
22 JUL 00	(Prince Of India		184.96	2320.45
23 JUL 00	(TRF Savings - 292939	290.00		2610.45
24 JUL 00	(Prince Of India	183.54		2793.99
25 JUL 00	(DDB - 27273992	85.63		2879.62
26 JUL 00	(Prince Of India	261.72		3141.34
27 JUL 00	(Boots REF:293848	165.40		3306.74
28 JUL 00	(Carpet City REF:39484		230.75	3075.99
29 JUL 00	(CHO - 38282892		81.03	2994.96
Final Balance £				2094.35

Done My Computer

Hotpic

The Hotpic component is contained in the FORMHPIC component file and lets you create 'hot' areas in a remote form usually over a photograph, map, or an illustration. For example, you could have a map of the US with the various states defined as different hot areas; here is the remote form in design mode:



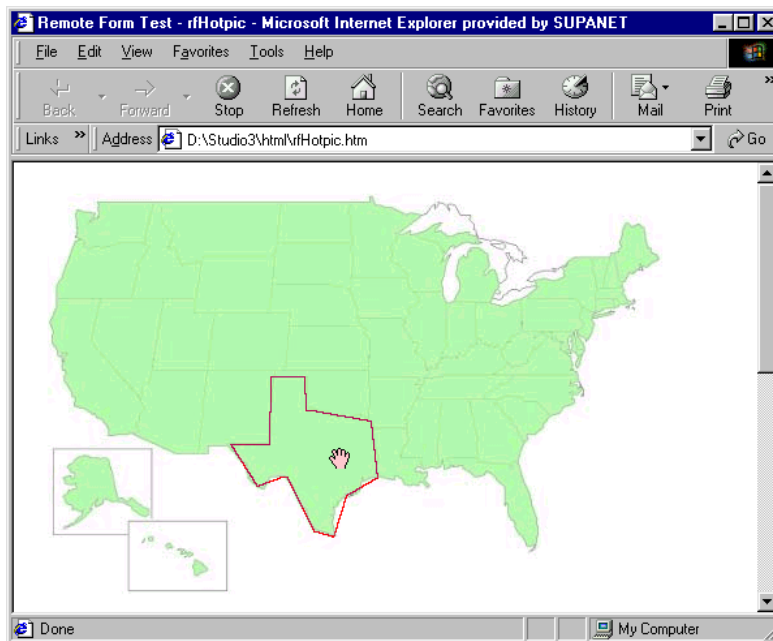
The Hotpic component appears on your remote form as a single rectangle placed over your whole image with separate hot areas defined within the rectangle. In design mode, you can use various mouse/key combinations to add new hot areas, add nodes to the current area, or move existing areas, as follows:

Add a hot area	Shift-Right click in the hotpic field; new hot area is added to the top-left of the hotpic component, you can move the new area, add nodes and reshape area as below.
Move area within hotpic	Right click inside the area and drag
Select an area	Right click inside the area; the Property Manager displays the properties for the selected area.
Delete hot area	Ctrl-Right click on the area
Add node to an area	Shift-Right click on a node; new node is added next to existing node
Move node to reshape area	Right click on node and drag
Delete node	Ctrl-Right click on node

When you Right-click inside a hot area within the hotpic field rectangle, the Property Manager shows the properties for the selected area. The \$currentid property specifies the id for the current hot area. As you add hot areas to the field in design mode, consecutive numbers are assigned to each hot area and in most cases you can use these numbers to identify an area; you can however change the ids or add your own default value for each area. Alternatively, you can assign a name to each hot area in the \$currentname property. At runtime, the \$arealist property contains a list of areas in the hotpic field.

You can assign a cursor icon to each area in the \$currentcursor property; the cursor is displayed when the user's mouse enters the area. You can also highlight an area in several ways: \$invertonenter inverts the area when the mouse enters the area; \$frameonenter highlights the area by displaying a frame; and \$flashonclick inverts when the user clicks the area.

The following screenshot shows the state of Texas being highlighted when the mouse is over that hot area. The \$frameonenter property is enabled and \$currentcursor is set to 5010 from the Hand Cursors page in the Omnispic icon data file. Note that you need to include the name of the page containing any cursor icons you have used in the \$iconpages property of the remote form.



The Hotpic component reports the evAreaClicked event which you can detect in the \$event() method for the hotpic component. The event passes the pAreaid and pAreaname parameters containing the id and name of the selected hot area. In the above example, pAreaname could be used to pass the name of the state selected, assuming 'Texas' has been added to \$currentname for that area.

Icon array

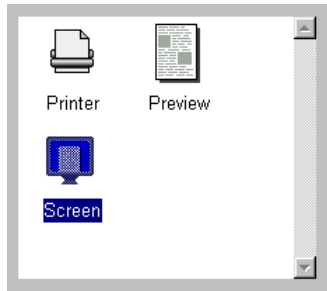
The Icon array component is contained in the FORMICON component file and lets you display a list of icons that the user can click on to select. The data for an icon array is supplied from a list specified in the component's \$dataname property. The first two columns of the list must contain the icon id and text label for each icon; subsequent columns can hold data that you read from the list when the user clicks on an icon. The icons to be displayed in the icon array can be from the Omnispic or Userpic icon data file, or the #ICONS system table in the current library. Remember that you need to include the name of the page containing your icons for the icon array in the \$iconpages property of the remote form.

In addition to the standard field properties, icon arrays have several properties that control the appearance and behavior of the component. The \$showtext property displays the text labels and is turned on by default. The icons in an icon array are normally 48x48 pixels by default, but you can display small 16x16 icons by enabling \$smallicons for the component; when in small icon mode you can limit the width of the text labels using \$smalltextwidth which must be at least 20 pixels. In addition you can add a grey button to each icon by enabling the \$buttonbackground property. The \$multipleselect property lets the user select more than one icon at a time.

You must define the list instance variable for an icon array with at least two columns, the first column for the icon id and the second column for the text label for the icon. The id numbers are those assigned to each icon in the Icon Editor and displayed in the Select Icon dialog. You can define and build the list in the \$construct() method of the remote form. For example:

```
; declare variable iIconId (Number 0dp)
; declare variable iIconName (Character)
; declare variable iArraylist of List type
; note the icons are from the PrintDest icon page in Omnispic
Do iArraylist.$define(iIconId,iIconName)
Do iArraylist.$add(1941,'Printer')
Do iArraylist.$add(1942,'Preview')
Do iArraylist.$add(1943,'Screen')
```

The above method produces the following icon array.



Icon arrays report the standard list events `evClick` and `evDoubleClick` which you can detect in the `$event()` method for the component. Both these events pass the `pLineNumber` parameter containing the line number of the list corresponding to the selected icon. The following `$event()` method for an icon array branches according to the value of `pLineNumber`.

```
On evClick
  Switch pLineNumber
    Case 1
      Do method $cinst.PrintToPrinter
    Case 2
      Do method $cinst.PrintToPreview
    Case 3
      Do method $cinst.PrintToScreen
  End Switch
```

Info

The Info or ‘cookie’ component is contained in the FORMINFO component file. It can store user information on the client machine between user sessions, and is a very useful component for increasing the user-friendliness of your web application. The Info component is not a real web-style cookie but allows similar functionality. The Info component is invisible in the form and should typically be placed outside the border of the form or behind another object.

The user information is stored in encrypted format on the client machine, and is retrieved automatically each time the user accesses your web application. The user information is stored in the Formcache folder which is located in the user’s Webclient folder under Windows, or the System folder on MacOS.

The Info component has to be instantiated in your remote form to receive events, so you must place it somewhere in your remote form, maybe behind another object or out-of-sight beyond the window border. The Info component reports the `evUserDataInit` event when the

object is instantiated, that is, when the remote form containing the object is opened on the client. Note you have to enable the `evUserDataInit` event in the `$events` property of the `Info` object. The `evUserDataInit` event passes the `pUserData` parameter containing the client's user information. If the client machine has never accessed the form before, the `pUserData` parameter is empty.

You can trap the `evUserDataInit` event in the `$event()` method of the `Info` object and check the contents of the `pUserData` parameter. To store the information, you need to assign a value to the `$userdata` property of the `Info` component. You need to write a method in your remote form to assign to `$userdata` in whatever format you wish; the format depends on what information you want to store, but this would typically be a username only, or maybe a username and password. In the latter case, you need to separate the username and password using something like '@: @' in the format 'username@: @password'. When you read the user info back in via the `pUserData` parameter, you should be able to indentify the separator and strip out the different parts of the user information.

The following example demonstrates a simple username logon form. It contains a page pane with 4 panes and the `Info` component is placed out-of-sight in the background of the form below the bottom edge of the form. The following screenshots show each pane in design mode.

Pane 1



Pane 2



Pane 3



Pane 4



When the form is instantiated on the client, the `Info` object receives the `evUserDataInit` event and its `$event()` method is called; note that while this method runs the first pane of the remote form is displayed showing the "Please Wait" message. The `$event()` method for the `Info` object is as follows:

```

; iLogonName is instance var of type Character
On evUserDataInit
    Calculate iLogonName as pUserData
    If len(iLogonName)    ;; there is a username, go to welcome pane
        Do $cinst.$objs.page.$currentpage.$assign(3)
        Do $cinst.$objs.displayname.$redraw()
    Else    ;; username is empty, go to user logon pane
        Do $cinst.$objs.page.$currentpage.$assign(2)
    End If

```

The logon screen (pane 2) contains an entry field (\$dataname iLogonName) and a Logon pushbutton. The method for the logon button is as follows:

```

On evClick
    Do $cinst.$objs.cookie.$userdata.$assign(iLogonName)
    Do $cinst.$objs.displayname2.$redraw()
    Do $cinst.$objs.page.$currentpage.$assign(4)
    ; username is stored and pane 4 is displayed

```

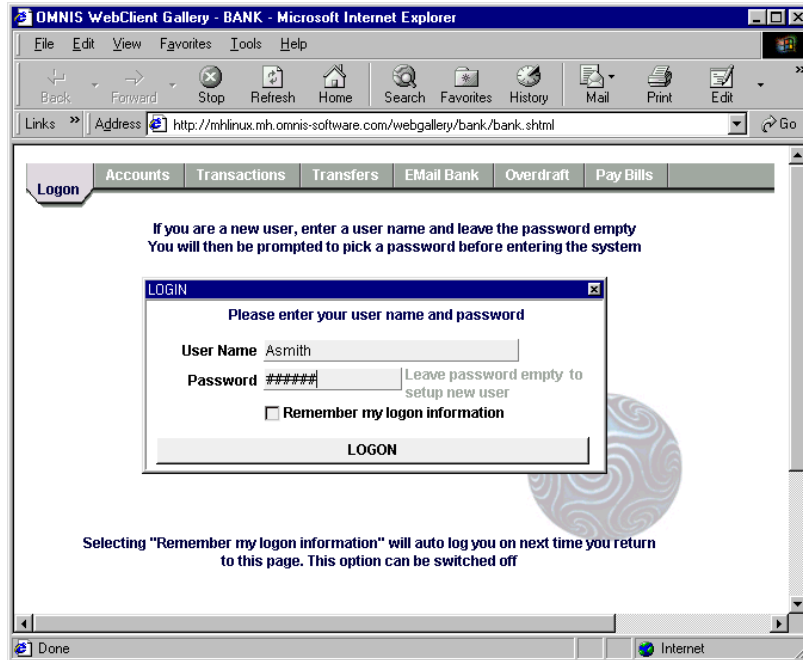
The Welcome back screen (pane 3) is displayed if/when the user returns to the form. This pane contains a display field showing the Username, and also provides a button to clear the current user information. The method for this button is:

```

On evClick
    Calculate iLogonName as ""
    Do $cinst.$objs.logon.$redraw()
    Do $cinst.$objs.cookie.$userdata.$assign("")
    Do $cinst.$objs.page.$currentpage.$assign(2)
    ; the current user data is cleared and the user is switched
    ; back to the Logon screen ie pane 2

```

The next example showing Info is from the Bank demo application that uses a Username and Password logon screen. When the user accesses the Bank example for the first time, they are prompted to enter a username and password, otherwise if they have used the application before, they are logged on automatically using the data stored on the client machine. The application uses a page pane with 9 panes; this is the logon screen:



The logon mechanism is similar to the first Info example, except that it stores a password as well as a username. When the user accesses the Bank remote form, the `evUserDataInit` event is sent to the Info object, which is placed in the form just below the bottom edge. The `$event()` method for the Info object is as follows:

```
; contains local vars tmp, tmp_user, tmp_password of Char type
; and tmp_sep of Number type
On evUserDataInit
    Calculate tmp as pUserData
    If len(tmp)    ;; there is a username
        Calculate tmp_sep as pos("@:@",tmp)
        ; finds the position of the separator
        Calculate tmp_user as mid(tmp,1,tmp_sep-1)
        ; extracts the username
        Calculate tmp_password as
            mid(tmp,tmp_sep+3,len(tmp)-tmp_sep+2)
            ; extracts the password
        Calculate iUserName as tmp_user    ;; assigns to instance vars
```

```

        Calculate iPassword as tmp_password
        Do method $checkuser (kTrue)
        ; checks if username exists: see below
    Else        ;; if pUserData is empty
        Do $cinst.$senddata(#NULL,iUserName,iPassword)
        Calculate iPassword as ""
        Do $cinst.$objs.page.$currentpage.$assign(2)
        ; go to new user pane in remote form
        Do $cinst.$objs.tab.$::currenttab.$assign(1)
        Do $cinst.$objs.username.$redraw()
    End If

```

The \$checkuser method determines whether or not the user information passed from the client is stored in the database, and if not the user is directed to the logon screen on pane 2 of the page pane.

```

; $checkuser method
; contains pIsAutoLogon parameter (boolean)
Do $cinst.$senddata(#NULL)
Single file find on fUsers.username (Exact match) {iUserName}
; searches in the database for user in iUserName
If flag true
    Do $cinst.$senddata(iUserName)
    Do $cinst.$objs.current_user.$redraw()
    If pIsAutoLogon
        Calculate $cinst.$objs.autologon.$visible as kTrue
    End If
    Do method $buildaccountlist    ;; get account info
    Do method $show_accounts
    ; $show_accounts displays the user's Accounts in pane 3
    Calculate iLoggedOn as kTrue
    If iRemember        ;; auto logon
        Calculate $cinst.$objs.cookie.$userdata
            as con(iUserName,"@:@",iPassword)
    End If
Else
    Do $cinst.$redraw()
    Calculate iPassword as ""
    Calculate iRemember as kFalse
    Do $cinst.$senddata(iPassword,iRemember)
    Do $cinst.$objs.page.$currentpage.$assign(2)    ;; Logon pane
    Do $cinst.$objs.tab.$::currenttab.$assign(1)
    Do $cinst.$showmessage(con("Sorry, we cannot find user
        ",iUserName," in our database. Please try again"),"Error")

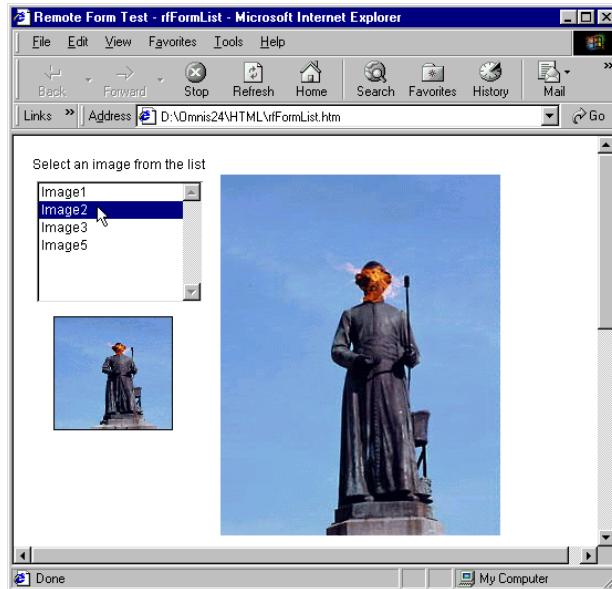
```


End If

Jpeg

The Jpeg form component is contained in the FORMJPEG component file, and lets you display Jpeg graphics data from an Omnis or server database. The image data is held in an instance variable of Picture type specified in the \$dataname property. The Jpeg form component reports no special events.

The following example form displays Jpeg files chosen from a list. When the remote form is instantiated, a list of images is built and shown in the headed list. The form contains three instance variables: iImageList (List), iImageName (Char), and iImage (Picture). The form actually contains a large jpeg component and a small one to show a thumbnail image, but both fields are assigned the variable iImage.



```
; $construct() method of the form
Set main file {fImages}
Do iImageList.$define(fImages.Name)
Set current list iImageList
Build list from file
Do iImageList.$redefine(iImageName)
```

The \$construct() method in the form builds the list of images, but the \$event() method behind the headed list loads the selected image. The method loads the image from the Omnis database and assigns it to the instance variable; the large image and the thumbnail are then redrawn while the image data is sent to the client using the \$senddata() method.

```
; $event() method for the headed list
On evClick
  Set main file {fImages}
  Do iImageList.$loadcols()
  Single file find on fImages.Name (Exact match) {iImageName}
  If flag true
    Calculate iImage as fImages.Image
    Do $cinst.$objs.thumb.$redraw()
    Do $cinst.$objs.bigimage.$redraw()
    Do $cinst.$senddata(#NULL,iImage)
  End If
```

The \$noscale property specifies whether or not the image is scaled; when set to kTrue, the true height, width, and proportions of the image are maintained, when kFalse, the image is scaled to fit the size and shape of the component itself. The Jpeg component has one or two additional properties under the Custom tab in the Property Manager; some of these you can set in design mode, while others are runtime properties. When set to kTrue, \$palette specifies that the image uses the color palette stored with the image. \$imageheight and \$imagewidth contain the height and width of the current image data. \$fast specifies that faster though less accurate processing of the image occurs. \$nosmooth disables smoothing of the image; when set to kFalse, smoothing occurs which smooths hard edges in the image and decreases the file size. When set to kTrue, the \$allowclipboard property lets the user paste an image into the field from the clipboard on the client.

The \$writejpeg(cFilename) method writes a Jpeg file to the client from the current image data with the filename and path specified in cFilename. The \$writejpeg() method must be executed on the client.

Label

The Label component is contained in the FORMBACK component file, and is installed along with the basic field components in the Omnis web client installer. It is located under WEB Background Objects group in the Component Store.

The Label object lets you place text labels in your remote forms, for example, to label entry fields. The text for the label is contained in the \$text property. The label component has the \$fieldstyle property which is set to OMNISweb by default. You can set the style to None and assign your own font and style properties. You can set the color of the text in \$textcolor.

The label object is a background component which means when you use the notation you have to reference the object using its ident, for example, the following notation changes the text for a label which has the ident 1002.

```
Do $cinst.$objs.1002.$text.$assign('NEW TEXT')
```

List

The single-column list component is contained in the FORMFLDS component file, and is installed on the client by the default Omnis web client installer.

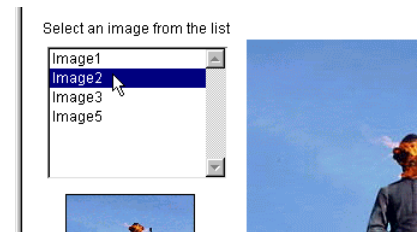
Lists are useful for displaying short single columns of data that allow the user to make a selection. The content of the list is supplied from an instance variable specified in the \$dataname property. Lists report the evClick and evDoubleClick events which you can detect in the \$event() method for the object.

Both these events pass the parameter pLineNumber containing the line number of the list row clicked.

The following example uses a simple list box to allow the user to select a picture to display in their browser; the example on our website uses a single column headed list but the implementation is exactly the same.

The \$event() method for the list detects which line is selected and loads the appropriate picture from the database. The instance variable for the list field is called iImageList and has a single column, iImageName, containing the names of the images.

```
; $event() method for list
On evClick
  Set main file {fImages}
  Do iImageList.$loadcols()
  Single file find on fImages.Name (Exact match) {iImageName}
  If flag true
    Calculate iImage as fImages.Image
    Do $cinst.$objs.thumb.$redraw()
    Do $cinst.$objs.bigimage.$redraw()
  End If
```



You could build the list data in the \$construct() method of the remote form; in this example method, the images are stored in an Omnis database, but the data could equally be extracted from a remote server database.

```

; $construct() method of form
Set main file {fImages}
Do iImageList.$define(fImages.Name)
Set current list iImageList
Build list from file
Do iImageList.$redefine(iImageName)

```

Marquee

The Marquee component is contained in the FORMMARQ component file and lets you display continuously scrolling text areas in your remote forms; you could use Marquee for news headlines or stock prices, or anything that needs to grab the user's attention.

[Click here for latest Omnis news](#)

You can enter the text for marquee object in the \$message property. You can use the style() function to embed icons and colors in the scrolling text message.

You can set the text and background color using \$textcolor and \$backcolor, and set up the font using \$font and \$fontsize. You can set the \$speed of the scrolling message (the lower the value, the slower the scrolling) and you can set the \$steps which controls how much the message jumps while scrolling. You can scroll the marquee in the opposite direction by specifying a negative value for \$steps.

Page pane

The Page pane component is contained in the ORFCMAIN web client, and is installed on the client by the default Omnis web client installer.

Page panes have a number of panes in which you can place other fields or components, and at any one time only one pane is visible. They allow you to display a number of different 'screens' or views in the same remote form. For example, in a banking application, you could use separate panes of a page pane to display the fields and options for different accounts, transfers, personal account details, and so on. Several of the web client examples on www.omnis.net use page panes.

The \$pagecount property determines how many panes the field has, and the \$currentpage property specifies which pane is currently visible. In design mode, you cannot click on the separate panes, rather you need to set \$currentpage in the Property Manager to make a pane visible to change it or add fields. At runtime on the client, you can set the \$currentpage property to display a particular pane. For example, the following method could be placed behind a button, button area, or adapted for use with a tab bar component.

```

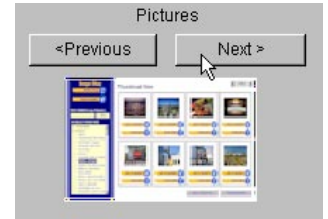
On evClick
  Do $cinst.$objs.pagepane.$currentpage.$assign(2)

```

Picture

The Picture component is contained in the FORMFLDS component file, and is installed on the client by the default Omnis web client installer.

The picture component can display image data retrieved from either a server database or an Omnis data file; the latter supports the new true color shared picture mode. The component requires an instance variable of Picture type specified in the \$dataname property. Pictures can have a horizontal and/or vertical scrollbar and scaling is controlled using the \$noscale property. If set to kTrue, the \$cachepicture property enables the client to store a copy of the image suitable for drawing without conversion, which provides faster drawing on the client, but more memory is used.



In addition to the general field events evBefore and evAfter, pictures respond to evClick and evDoubleClick events which you can detect in the \$event() method for the component.

The following example methods implement a simple picture database (such as the one shown) with a Next and Previous button allowing you to view the images stored in an Omnis database. The picture component in the form is assigned an instance variable called iPicture; the picture field itself requires no methods, all the logic is built into the pushbuttons.

```
; $event() for the Next pushbutton
On evClick
  Set main file {fImages}
  Next                ;; or Previous for previous button
  If flag false
    Find first        ;; or Find Lasr for previous button
  End If
  Calculate iPicture as fImages.image
  Do $cinst.$objs.picture.$redraw()
  Do $cinst.$senddata(#NULL,iPicture)
```

Port

The Port component is a non-visual component that lets you communicate asynchronously with serial devices that are connected to a port on the client machine, for example, a chip card reader connected to a COM port on the client. It is contained in the FORMPORT component file. Note the Port component does not interface with socket ports.

The Port component uses internal timers to read from and write to the port on the client after a specified amount of time has expired. Support is provided for streaming data back to the web client server either in blocks or as one block when reading is complete. Multiple port support is also provided. This lets you transmit and receive on several communications ports on the client at the same time. The settings for each port are based on the settings in the port component properties. At runtime, the Port component maintains a separate list of runtime properties for each port. This allows you to have multiple ports, each with its own configuration.

The Port component does not provide any methods and is not associated with any data variables in Omnis. All actions are controlled by properties and events. In order to perform an action such as reading a file, you assign a value to the action property (\$paction) and a corresponding event containing the data read is passed when the operation is complete. The following actions are available; these are described in greater detail in subsequent sections.

kGetPortNames	Triggers the evPortNames event which returns a list of portnames.
kOpenPort	Sets the state of the current port using the current Port Settings, opens the port and sends an event to indicate the success/failure of the operation.
kReadData	Reads data from the port.
kWriteData	Writes data to the port and sends an evDataWritten event when the operation is completed.
kClosePort	Closes the port and sends an event to indicate the success/failure of the operation.
kPauseRead	Pauses the read operation for the current port. Executing another kReadData for the current port resumes the read operation.
kCancelRead	Terminates the current read and sends an evDataRead event together with any data that has been read thus far.
kPauseWrite	Pauses the write operation for the current port. Executing another kWriteData for the current port resumes the write operation.
kCancelWrite	Terminates the current write operation and sends an evDataWritten event.

The following events are available for the port component. Most of the events pass a portname. This can be used to determine which port the event occurred for in situations where more than one port is open at the same time. The web client does not handle simultaneous events. As a result, the following events will not actually be processed until the event that is currently being executed by the web client is completed.

- ❑ **evPortNames(pPortList)**
passed in response to assigning \$paction the kGetPortNames constant. The parameter pPortList contains a list of available ports for the platform that the client is currently running on. It should be noted that for both Windows & Linux, there is no current OS method for detecting available communications ports. As a result these platforms will

always pass a list of standard ports. However, the Mac is capable of listing available communications ports via the Comm Toolbox Api, including Comm Toolbox savvy serial devices that are connected via USB.

- ❑ **evDataReading**(pPortName, pData, pBytesRead)
passed in response to assigning \$paction the kReadData constant if read streaming is enabled. The parameter pData contains a binary variable which holds the data that has been read from the port. pBytesRead contains the number of bytes read so far.
- ❑ **evDataRead**(pPortName, pData, pBytesRead)
passed when the read operation is complete (i.e. when either a timeout occurs or a kCancelRead is executed). If data streaming is used, then any remaining data is returned at this point. If data streaming is not used, all data is returned at this point.
- ❑ **evDataWriting**(pPortName, pBytesWritten)
passed after \$writeblocksize bytes are written.
- ❑ **evDataWritten**(pPortName, pBytesWritten)
passed when a write operation is completed.
- ❑ **evPortOpened**(pPortName)
passed in response to assigning \$paction the kOpenPort constant.
- ❑ **evPortClosed**(pPortName)
passed when the port is closed.
- ❑ **evPortError**(pPortName, pError)
passed if an error occurs. The parameter pError passes the appropriate error constant.

Getting a list of Port Names

It's not a good idea to hard code port names into your application as these will vary from platform to platform. In order to avoid this, the Port component provides an action for getting port names (kGetPortNames). This action passes an event which has an Omnis list as one of its parameters (remember you need to enable any events in the \$events property of the port component). For example:

```
; Code to get port names
Do $cinst.$objs.formport.$paction.$assign(kGetPortNames)

; formport event handling code
On evPortNames
    Calculate mylist as pPortList
```

Creating a Port

Once you have a port name, you can assign it to the port component as follows; this causes the port component to create an internal record for the port.

```
Calculate myportname as mylist.[$mylist.$line].ports  
Do $cinst.$objs.formport.$pportname.$assign(myportname)
```

Opening a Port

Once a port has been created, the following code can be used to open the port.

```
Do $cinst.$objs.formport.$paction.$assign(kOpenPort)
```

The above also sets the parameters of the port to automatically reflect the current settings of the port component. If different settings are required at runtime, these should be set before the port is opened. For example:

```
Do $cinst.$objs.formport.$pbaudrate.$assign(k9600)  
Do $cinst.$objs.formport.$pparity.$assign(kPortNoParity)  
Do $cinst.$objs.formport.$pdatabits.$assign(kPort8DataBits)  
Do $cinst.$objs.formport.$pstopbits.$assign(kPort1StopBit)  
Do $cinst.$objs.formport.$phandshake.$assign(kPortHardware)  
Do $cinst.$objs.formport.$paction.$assign(kOpenPort)
```

The following code should be placed in the \$event() method of the form component.

```
; $event() for formport  
On evPortOpen  
    ; This event will be received if the port is opened successfully  
    ; do appropriate coding here
```

Reading from a port

Once a port has been opened, reading may begin. The information passed from a read operation is dependant on the values of the \$spreadstream and \$pdatastream properties. \$spreadstream enables evReadingData messages. By default, these messages pass the number of bytes read. However, if \$pdatastream is set to kTrue then evReadingData also passes the last data block read. The size of this block is determined by setting the property \$preadblocksize. Once a read is started, data is read from the port every time the \$preadtimeinterval expires. Some examples of reading from a port are shown below:

```
; To start reading..  
Do $cinst.$objs.formport.$paction.$assign(kReadData)
```

The following code should be placed in the \$event() method of the form component.


```

; formport event handling code for data streaming..
On evReadingData
    ; Append each data block to binary variable as it is received.
    Calculate mybinvar as bytecon(mybinvar,pPortData)
    Calculate mybytesread as pBytesRead
On evDataRead
    ; Append final data block
    Calculate mybinvar as bytecon(mybinvar,pPortData)
    Calculate mybytesread as pBytesRead

```

The following method is for the formport event handling code if data streaming is not used (i.e. if \$pdatastream is set to kFalse). For the best performance, client scripting should be enabled for the following code. This stops evReadingData messages from being sent back to the server. However, these messages can still be used locally in order to obtain bytes read information. This information could be linked to another component, such as a progress bar, to indicate to the user the current state of the read.

```

On evReadingData
    ; Data is not being received in blocks, so just calculate bytes
    ; read for display purposes.
    Calculate mybytesread as pBytesRead
On evDataRead
    ; All of the data read is passed in pPortData. PBytesRead
    contains the final byte count.
    Calculate mybinvar as pPortData
    Calculate mybytesread as pBytesRead
    ; As this code is executing on the client, we need to call
    ; a function that exists on the server
    ; in order to pass our information back to the server.
    ; $sendtoserver is a function which simply
    ; puts the value of mybinvar into a variable which exists
    ; on the server
    Do method $sendtoserver(mybinvar)

```

Since information being read from the port is binary data, there is no easy way to determine when a read is complete. The Port component provides two ways of dealing with this. First, a timeout property (\$ptimeout) can be set. This causes a read to complete if no data is read after the number of seconds specified in \$ptimeout. Second, the Port component provides the action kCancelRead. This action can be used in the following manner to stop the read process:

```

Do $cinst.$objs.formport.$paction.$assign(kCancelRead)

```

Both of these methods result in an evDataRead message being generated. Reading may also be paused by executing the following:

```
Do $cinst.$objs.formport.$paction.$assign(kPauseRead)
```

To resume reading at the current position, simply assign kReadData again.

NOTE: Hardware Handshaking should be used whenever possible as it ensures that the Port component is synchronised with your serial device and that no data is lost. If Hardware Handshaking is not used and you are reading a lot of data, then it is possible that some information may be lost as your serial device will continue to transmit continuously without regard to the state of the Port component.

Writing to a Port

Writing to a port works in a similar way to reading from a port with the exception that only bytes written information is passed with evWritingData and evDataWritten messages. If \$pwritestream is enabled, then an evWritingData message is generated every time \$pwriteblocksize bytes are written. When the write operation is complete, the evDataWritten message is generated. Some examples of writing to a port are shown below:

```
; Before writing to a port, the data to be written must be assigned
; to the port component.
Do $cinst.$objs.formport.$pdata.$assign(mybinvar)
; Once data has been assigned, the write process can be
; started as follows:
Do $cinst.$objs.formport.$paction.$assign(kWriteData)
```

The following code should be placed in the \$event() method of the port component:

```
On evWritingData
    ; Get current byte count
    Calculate mybyteswritten as pBytesWritten
On evDataWritten
    ; Get final byte count
    Calculate mybyteswritten as pBytesWritten
```

In addition to the above, the following write actions are also available:

```
; To cancel a write operation before it is completed and generate an
    evDataWritten message.
Do $cinst.$objs.formport.$paction.$assign(kCancelWrite)
; To pause the write process. (May be resumed by assigning the
    kWriteData action)
Do $cinst.$objs.formport.$paction.$assign(kPauseWrite)
```

Closing the Port

The following can be used to close the port.

```
Do $cinst.$objs.formport.$paction.$assign(kClosePort)
```

This will stop all reading and writing processes for the port and will discard any information that is currently in the buffer for the port. If you are reading information, you should use the `kCancelRead` option to ensure that all data is returned before closing the port. The following code should be placed in the `$event()` method of the `formport` component.

```
; formport event handling code
On evPortClosed
    ; This event will be received if the port is closed successfully
    ; do appropriate coding here.
```

Multi Port Support

In order to support multiple communications ports at the same time, the `evPortError`, `evPortOpened`, `evPortClosed`, `evReadingData`, `evDataRead`, `evWritingData` and `evDataWritten` events all pass the parameter `pPortName`. This can be used to determine which port the event occurred for.

Each action is executed on the current port. The current port is whatever you assigned to `$pportname`. You may freely move between ports by assigning a port name to `$pportname` and execute actions on the port. Each port also has its own set of configuration settings at runtime. These settings are derived from the default settings that you set in the component properties in design mode. When you assign a property at runtime, the value is assigned to the property of the current port. For example:

```
; The following example sets different read intervals for ports
COM1: and COM2:
Do $cinst.$objs.formport.$pportname.$assign("COM1:")
Do $cinst.$objs.formport.$preadinterval.$assign(200)
; now COM2
Do $cinst.$objs.formport.$pportname.$assign("COM2:")
Do $cinst.$objs.formport.$preadinterval.$assign(100)
```

Aborting Read/Write Operations

Although the port component is asynchronous, the Omnis Server does not handle messages in an asynchronous manner. As a result, if you use data streaming with a small block size, you will find it difficult to click other web client components such as buttons because web client will be too busy processing `evReadingData` events to process `evClick` events. This means that if a problem does occur during data transfer, the user will find it difficult to stop the transfer under these conditions. In order to remedy this, the port component provides the property `$pinterruptkey`. If this property is set to `kTrue` then all keyboard input is rerouted to the port component. Once this property has been set, holding down the "END" key will terminate reading and writing for all ports and control is returned to the client.

Error Handling

All errors are passed with the `evPortError` event. The port name is passed together with the error code to indicate the error that occurred for that port. The following errors are passed:

<code>kPortOpenError</code>	passed if an error occurred when the port was opened.
<code>kPortCloseError</code>	passed if an error occurred when the port was closed.
<code>kPortConfigError</code>	passed if invalid parameters were specified for the port.
<code>kPortOutOfMemoryError</code>	passed if there is not enough available memory for the operation.
<code>kPortReadError</code>	passed if an error occurred during a read process.
<code>kPortWriteError</code>	passed if an error occurred during a write process.
<code>kportNoDataError</code>	passed if a <code>kWriteData</code> is executed before data is assigned to <code>\$pdata</code>

Print

The Print component is contained in the FORMPRI component file, and lets you print a report on the client machine. The Print component is a non-visual component, hence it is invisible in the form and should typically be placed outside the border of the form or behind another object. The Print component can send an Omnis report, which was previously printed to memory on the Omnis Server, to either the Screen, Preview, or Printer on the client machine (note the Printer destination is not supported on Linux clients). This allows you to create all types of report on the client that would not be possible with Html alone. For example, in a shopping application you could generate a graphically rich order confirmation and print it to the screen or preview; the user could then print the order to their printer connected to the client. Note that images with a high definition may lose their high definition when printed to the memory device on the server and will therefore suffer a loss of quality when printed on the client.

To print a report to the client you first need to create a method on the Omnis Server that prints your report to the `kDevMemory` printing device. Next you need to assign the resulting binary report data to the Print component on the remote form, and assign a destination to the `$action` property.

You can direct the report to the Screen, Preview, or Printer device by assigning the appropriate constant to the `$action` property; note this is a runtime property only. When this property is assigned it sends the current report data to the specified destination on the client machine. The report data must have been previously assigned to `$reportdata`. This property can be assigned one of the following constants

kFPriNone	The report is hidden.
kFPriSendToPreview	The report is shown on screen in Page Preview mode.
kFPriSendToPrinter	The report is printed to the current local or network Printer; the user will be prompted by the Job Setup dialog prior to printing.
kFPriSendToScreen	The report is shown on Screen.

The \$reportdata property takes the binary representation of an Omnis report, which was previously printed to the memory printing device; note this is a runtime property only. The following method assigns the report data in 'myRepData' to the print component in the current remote form instance:

```
Do $cinst.$objs.RepField.$reportdata.$assign(myRepData)
```

The \$reporttitle property lets you assign a title to the document when printed to printer. The \$zoomon property specifies the initial zoom state for page preview mode. When this property is true, the report is shown at 100%. In addition, \$showtoolbar and \$showstatusbar let you display the toolbar and statusbar in screen or preview mode, while \$showhscroll and \$showvscroll let you enable the horizontal and vertical scrollbar.

You can change the text on the Job Setup dialog, the Toolbar buttons, and the Status bar using the \$strjobdlg..., \$strbutt..., and \$strstatus properties, respectively. For all these properties you can turn on the Help Tips in the Property Manager and move the mouse over the property name to show the original text. When these properties are empty, the text from the component's resources is used.

The following method sets the current print device to memory, sets the report name, sets the paper size and orientation, prints the data to memory, and assigns the resulting data to the print component (called RepField) on the remote form.

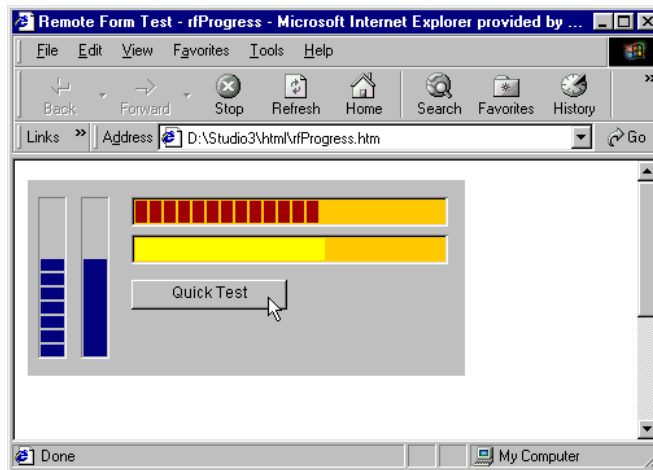
```
; Prepare the memory device
Do $cdevice.$assign(kDevMemory)
Do $root.$prefs.$reportdataname.$assign(myRepData)
If flag true
    ; set required paper size (optional)
    Do $root.$prefs.$paper.$assign(kPaA4)
    Do $root.$prefs.$orientation.$assign(kOrientPortrait)
    ; print the report
    Set report name {myReport}
    Print report
    ; assign report binary to report field
    Do $cinst.$objs.RepField.$reportdata.$assign(myRepData)
    ; print report to printer on client machine
    Do $cinst.$objs.RepField.$action.$assign(kFPriSendToPrinter)
End if
```

Progress

The Progress component is contained in the FORMPROG component file and lets you display a progress bar in your remote forms. The value of the progress is specified in the \$val property which is typically linked to the value of a counter in a looping command. You can specify the range for the progress bar in the \$min and \$max properties. You can specify the \$backcolor of the bar as well as the \$progresscolor. The current value of the progress bar can be either a series of blocks (when \$blocks is kTrue) or a continuous strip. Furthermore your progress bar can be either vertical or horizontal by setting the \$vertical property.

Note that the progress bar component has no events or built-in methods of its own. Rather you control it by assigning to the \$val property at runtime on the client machine.

The following example form contains some progress bars that are activated when a pushbutton is clicked.



The remote form has four progress bars named prog1 to prog4 and a simple looping method behind the pushbutton which activates the progress bars.

```
; $event() for button
On evClick
  For loop from 1 to 100 step 1
    Do method $setprog (loop)
  End For
  For loop from 100 to 1 step -1
    Do method $setprog (loop)
  End For
```

The \$event() method for the button steps from 1 to 100 and back to 1 sending the current value in the loop parameter to the \$setprog class method which in turn assigns the current value to the \$val property for each progress bar.

```
; $setprog() method
; contains pValue parameter
Do $cinst.$objs.prog1.$val.$assign(pValue)
Do $cinst.$objs.prog2.$val.$assign(pValue)
Do $cinst.$objs.prog3.$val.$assign(pValue)
Do $cinst.$objs.prog4.$val.$assign(pValue)
```

Note that both the \$event() and \$setprog methods are set to execute on the client which ensures, that while the loop is running, no calls are made back to the server and the progress bars are displayed as intended. If these methods were executed on the server, there would be a delay on the client while the loops were running and the effect of the progress bars would be lost.

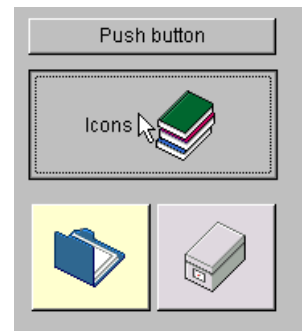
Pushbutton and Button Area

The Pushbutton and Button area components are contained in the FORMFLDS component file, and are installed on the client in the default Omnis web client installer.

Pushbuttons and button areas let you run a method in response to a user click and are therefore suitable for triggering an action on the client, maybe to confirm some user input or to select an option. Button areas behave very much like standard pushbuttons except that they are invisible, so you can place them over your own button image, graphic logo or icon. Buttons report the evClick event which you can detect in the \$event() method for the button. Note that you need to enable the evClick event in the \$events property of the button for it to be reported on the client.

You can set the text for a pushbutton in design mode in the \$text property. You can specify the look-and-feel of a pushbutton using the \$effect and \$buttonstyle properties, and you can add an icon using the \$iconid property. Note that pushbuttons do not have a \$dataname.

A useful property of pushbuttons and button areas is \$cursor which lets you assign a cursor icon to the object; when the user's mouse enters the button, the cursor changes to the specified icon. Remember that you need to include the name of the page containing your cursor icon in the \$iconpages property of the remote form.



The following example method detects the evClick event and runs a method to test if a password field is either in the correct format or if the password exists and is correct.

```
On evClick
  If len(iPassword)<4
    Do $cinst.$senddata(#NULL)
    Do $cinst.$showmessage("You must enter at least 4
      characters in your password","Error")
    Quit method
  End If
  If iPassConfirm<>iPassword
    Do $cinst.$senddata(#NULL)
    Do $cinst.$showmessage("You password and confirmation
      do not match - Try again","Error")
    Quit method
  End If
  Do $cinst.$senddata(#NULL)
  Do $cinst.$showmessage("Thankyou for registering with
    this example - Please remember your username and
    password for next time.", "OK")
  Do method $setupnewuser
```

Like pushbuttons, button areas report the evClick event, so whatever code works for a pushbutton will work for a button area. For example, the following code responds to a click and changes the current page of a page pane and a tabbar in the remote form.

```
On evClick
  Do $cinst.$objs.pagepane.$currentpage.$assign(2)
  Do $cinst.$objs.tabbar.$::currenttab.$assign(1)
```

Quicktime

The Quicktime or 'Movie Player' component is contained in the FORMQT3 component file and lets you display Quicktime version 3 and 4 movies and sound files in your remote forms. If the client does not have Quicktime 3 or 4 installed, then a Quicktime "Get4" logo displayed.

The component has many properties which you can view and set in the Property Manager. If you are unsure what a property does, show the Help tips for the Property Manager (Right/Ctrl-click and select Help Tips option) and move your mouse over the property to display its description.

The \$moviefile property lets you specify the name and path of a movie or sound file. The \$movieurl property lets you specify the URL for a movie or sound file. Movies are streamed from the web server, thus removing the need for the file to be completely downloaded

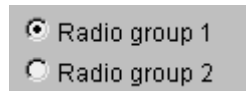
before viewing. Note you can only assign moviefile or movieurl, not both. Assigning one will automatically clear the other one.

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

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

Radio group

The Radio group component is contained in the FORMFLDS component file, and is installed on the client by the default Omnis web client installer.



The Radio group or radio button component presents 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 enter the text to display to the right of the radio buttons by entering a comma-separated list in the \$text property for the radio group. You can specify the arrangement of the radio buttons in the group using the \$horizontal and \$columncount properties.

The instance variable you associate with a radio button group should be numeric. You enter the range of values for the radio group by entering values in the \$minvalue and \$maxvalue properties. At runtime, the instance variable assigned to the radio group is set to the value corresponding to the radio button selected and according to the range of values you specified.

Note that the images for a radio button on and off states (i.e. the circle with and without the dot) are stored in the Omnispic icon data file in the 'Multistate 1' page (Multistate 3 for MacOS) and for these to display on the client you have to add this icon page to the \$iconpages property for your remote form.

Radio buttons report the evClick event, but no parameters are passed. You can detect the evClick event in the \$event() method for the object.

Roll button

The Roll button component is located in the FORMROLL component file. The Roll button is a graphical pushbutton that highlights when the user passes their mouse over the button. Roll button has a number of properties to control the look and behavior of the object when the mouse is placed over it. You can specify the image (\$outsideimage and \$insideimage) and text (\$outsidetext and \$insidetext) to be displayed when the mouse is either inside (over) or outside (not over) the roll button. You can also specify the text offset using the \$textx and \$texty properties, and set the spacing of multi-line text using \$betweenlines.

Roll button reports the evIsInside and evClick events, which have to be enabled in the \$events property for the object.

Sidebar

The Sidebar component is contained in the FORMSBAR component file and provides a vertical bar from which the user can select groups and items. The content and setup of the sidebar is specified using an Omnis list variable specified in the \$dataname property of the sidebar. To position a Sidebar component in the remote form, you can set its \$edgefloat property; for example, you can set it to kEFposnLeftToolBar to “glue” the sidebar to the left side of the form.

The content of the Sidebar list can be built on the fly or loaded from a database as the remote form is instantiated. The list must contain lines that define the group names and the items in the group, in the following format:

Line #	List columns
1	Group 1 name
2	Item 1 name, IconID, Value
3	Group 2 name
4	Item 1 name, IconID, Value
5	Item 2 name, IconID, Value
6	Group 3 name
7	Item 1 name, IconID, Value
8	Item 2 name, IconID, Value
9	Item 3 name, IconID, Value

You can create and build the list in the \$construct() method of the form using a method similar to the following. The list iSidebarList is an instance variable in the form, along with the column variables iGroup, iIconID, iName, iID. The iID variable is used in other methods in the form to pass which item in the sidebar is chosen.

```

; $construct of remote form
Do iSidebarList.$define(iGroup,iIconID,iName,iID)
; the remainder of method builds content of sidebar list
Do iSidebarList.$add("Yellow",0,"") ;; Group 1
Do iSidebarList.$add("Yellow",k32x32+1,"Yellow",1)
Do iSidebarList.$add("Red",0,"") ;; Group 2
Do iSidebarList.$add("Red",k32x32+2,"Red",2)
Do iSidebarList.$add("Green",0,"") ;; Group3
Do iSidebarList.$add("Green",k32x32+3,"Green",3)

```

The sidebar component reports the `evIconPicked` event which passes the `pLinenum` parameter containing the line number in the list corresponding to the item selected. When considering the value of `pLinenum` you have to take account of all groups and items in the list. Using the list above `pLinenum` will be value 2 (ie line 2 in the list) for the first item in the group 1, value 4 for the item in group 2 and value 6 for the item in the third group.

You can detect the `evIconPicked` event in the `$event()` method for the sidebar and branch your method according to the item selected. The following method

```

; $event method for sidebar component
On evIconPicked
  Do iSidebarList.$line.$assign(pLinenum)
  ; selects the list line according to item selected in sidebar
  Do iSidebarList.$loadcols()
  ; loads the values in the selected list line, including iID
  Switch iID ;; branches according to value of iID
    Case 1
      Do something..
    Case 2
      Do something..
    Case 3
      Do something..
  End Switch

```

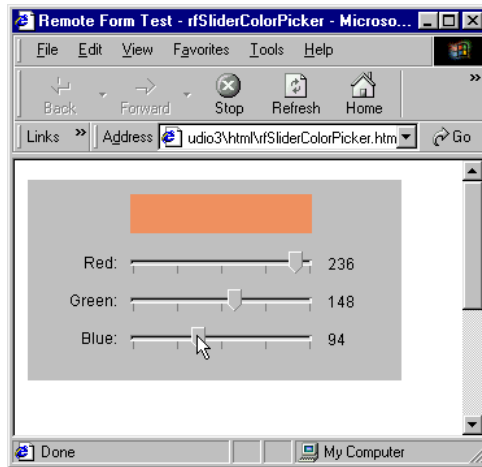
Slider

The Slider component is contained in the `FORMSLID` component file and provides a graphical thumb component that the user can drag to control the numeric setting of another component in your form, e.g. a volume control, progress bar, or a setting of some kind.

The current value of the slider is specified in the `$val`; at design time you can enter a default value, and at runtime `$val` holds the current value according to where the slider is positioned. You can specify the range for the slider in the `$min` and `$max` properties. Most of the other properties are self-explanatory and handle the appearance of the slider

component. The Slider reports 3 events: `evNewValue`, `evStartSlider`, and `evEndSlider` which you can detect in the `$event()` method for the component. These events all pass the current value of the Slider in the `pNewVal` parameter. As the user drags the Slider thumb the `evNewValue` event is triggered and `pNewVal` is sent to the `$event()` for the Slider.

The following example remote form has 3 sliders that let the user choose a color by setting the Red, Green, and Blue value of a field that is used to display the color. Each slider has the following properties set: `$min=0`, `$max=255`, `$val=0`, `$markfreq=64`, `$bigrange=kFalse`. The form contains 3 instance variables `iRed`, `iGreen`, and `iBlue` for the current Red, Green, Blue value, and next to each slider there is a display field showing the current value of the appropriate slider.

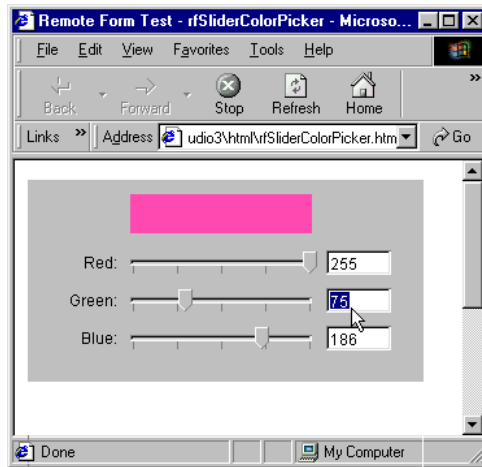


```
; $event() behind RedSlider
On evNewValue
    Calculate iRed as pNewVal
    Do $cinst.$objs.Red.$redraw()
    Do method $setcolor
```

The `$event()` methods for the green and blue sliders are almost identical except that they act on their respective color variables and display fields. The color of the display field in the form is set using a class method `$setcolor` as follows:

```
; $setcolor class method
; CurrentColor is name of the display field
Do $cinst.$objs.CurrentColor.$backcolor.
    $assign(rgb(iRed,iGreen,iBlue))
```

As a further refinement to the above example, you could make the Red, Green, and Blue fields enterable to allow the user to choose the color either using the slider or by entering a value.



The only code you need to add is behind each of the entry fields; note that you need to set the value of the slider to the value the user enters into each field, as well as limiting the value entered to 255. For example the \$event() method for the Red field is as follows:

```
On evAfter
  If iRed>255
    Calculate iRed as 255
    Do $cinst.$objs.Red.$redraw()
  End If
  Do $cinst.$objs.RedSlider.$val.$assign(iRed)
  Do method $setcolor
```

Again, the \$event() methods for the green and blue entry fields are almost identical except that they act on their respective color variables and entry fields. Note that for the slider and entry fields you must enable their events (evNewVal and evAfter respectively) in the \$events property for each object. Also note that you can make all the above methods execute on the client, since method execution does not need to pass back to the server for any of this example to work correctly.

Tab bar

The tab bar component is contained in the FORMTBAR component file and has a series of thumb tabs which the user can click on. The tab bar is typically used with the page pane component to present a form with a number of layers or pages, for example, a preferences or options form could have separate pages for different options accessed using a tab bar.

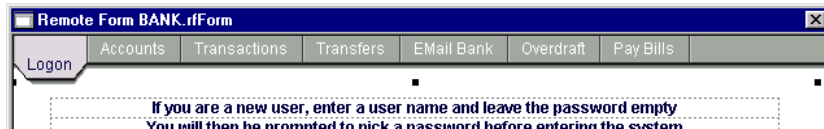
The tab bar reports the evClick event and passes the pLineNumber parameter containing the number of the tab clicked on. The following method is the \$event() method for a tab bar on

a window containing a page pane. The method detects the evClick on the tab bar and sets the currentpage of the page pane.

On evClick

```
Do $cinst.$objs.pagepane.$currentpage.$assign(pLineNumber)
; pLineNumber contains the number of the tab clicked
```

The next example shows an on-line banking form that uses a tab bar and page pane to organize the different options available to customers, such as Accounts, Transactions, etc.



The \$event() method for the tab bar detects the click and passes the number of the tab in the pLineNumber parameter. The method first tests whether or not the user is logged on and if not switches to the logon page. Next, the method uses a Switch statement to branch according to the tab number passed in the pLineNumber parameter.

On evClick

```
Do $cinst.$senddata(#NULL)
If not(iLoggedIn)
Do $cinst.$showmessage("Sorry, you must logon first","Error")
Do $cinst.$objs.tab.$::currenttab.$assign(1)
Quit method
End If
Switch pLineNumber
Case 1
Calculate iLoggedIn as kFalse
Calculate iUserName as ""
Calculate iPassword as ""
Do $cinst.$objs.page.$currentpage.$assign(2)
Do $cinst.$senddata(iUserName,iPassword)
Do $cinst.$redraw()
Case 2
Do $cinst.$objs.page.$currentpage.$assign(3)
Case 3
Do method $showoneaccount
Case 4
Do $cinst.$objs.page.$currentpage.$assign(5)
Case 5
Do $cinst.$objs.page.$currentpage.$assign(6)
Case 6
Do $cinst.$objs.page.$currentpage.$assign(7)
```

```

Case 7
    Do $cinst.$objs.page.$currentpage.$assign(8)
End Switch

```

Note the \$::currenttab property has two colons in its name to distinguish it from the internal property of the same. See [Web Component Properties](#) for more details.

Tile

The Tile component is contained in the FORMTILE component file and lets you display an icon repeated over the area it occupies in your remote form. You can specify the iconid and background color of the object. The icon for the tile object can be from the #ICONS system table in the current library, or the Omnispic or Userpic icon data file.

The tile object is a background component which means when you use the notation you have to reference the object using its ident, for example, the following notation assigns an icon to the tile component which has the ident 1055

```
Do $cinst.$objs.1055.$::iconid.$assign(k32x32+2185)
```

Timer

The Timer form component is located in the FORMTIME component file. It triggers an evTimer event after a specified time and runs the associated \$event method. You can specify a \$timervalue, which is interpreted using the setting of \$usesseconds which should be kTrue for seconds (the default) or kFalse for milliseconds.

The following example makes a very effective rolling demo-type presentation where the fadepic component is used to display a set of images and the timer is used to show the next image after a short delay.

```

; $construct() method of form
Open data file (Do not close other data) {[sys(10)],cardata}
Set main file {fCars}
Find first
Calculate iPicture as fCars.carPict
Do $cinst.$senddata(iPicture)
Do $cinst.$objs.formtimer.$running.$assign(kTrue)

```

```

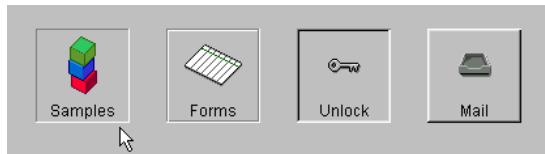
; $event() method for formtimer object
On evTimer
    Next
    If flag false
        Find first
    End If
    Calculate iFadeStyle as randintrng(1,35)
    Do $cinst.$objs.formfade.$fadestyle.$assign(iFadeStyle)
    Calculate iPicture as fCars.carPict
    Do $cinst.$senddata(iPicture)
    Do $cinst.$objs.formfade.$redraw()

; $event() method for formfade object
On evFadeFinished
    Do $cinst.$objs.formtimer.$running.$assign(kTrue)

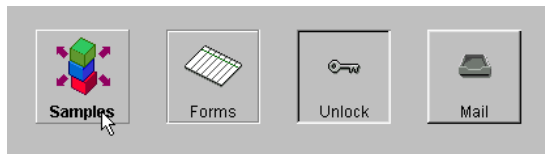
```

Transbutton

The Transbutton component is contained in the FORMTRAN component file and provides a standard button that displays a different image when the user's mouse is over the button. The transbutton component reports the evClick event which you can detect in the \$event() method for the object; in this respect a transbutton behaves like a standard pushbutton.



The Samples button when mouse OUTside



The Samples button when mouse INside

You specify the icons for the OUTside and INside images in the \$outsideicon and \$insideicon properties. Remember you must add the name of the icon page containing the icons for your transbuttons in the \$iconpages property of the remote form.

The \$boldover property specifies that the text is boldened when the user's mouse is over the transbutton, and the \$effect button provides numerous styles for the border of the button.

Tree list

The Tree list component is contained in the FORMTREE component file and provides a number of choices via an expandable selection of nodes arranged in a hierarchical list. The content and setup of the tree list is specified using an Omnis list variable specified in the \$dataname property of the tree list. 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. Note that you need to add the page name of any icons you use in your tree list to the \$iconpages property for your remote form; for the node expand (+) and collapse (-) icons in a tree list you can use the 'Multistate 2' page, while the 'Browsers' page provides several suitable icons.

The contents of the list can be built on the fly as the remote form is instantiated or loaded from memory. The list must contain lines that define the node names, icons, and the items in each group, as well as booleans for the \$enterable, \$expandcollapse, and \$textcolor properties. The \$expandcollapse value can be the sum of the constants kTREENodeExpCollapseAlways and kTREENodeExpanded. The list for the tree list should have the following structure:

Name	Name	\$iconid	\$ident	\$enterable	\$expand collapse	\$textcolor
RootNode			100	0	0	0
RootNode	Child 1	0	101	0	0	0
RootNode	Child 2	0	102	0	0	0
New Root		0	200	0	0	0
New Root	Child 1	0	201	0	0	0
New Root	Child 2	0	202	0	0	0

You can create and build the list in the \$construct() method of the form, using a method similar to the following. The list iTreeList is an instance variable in the form, along with the column variables iNodeRoot, iNodeChild, iNodeIcon, iNodeIdent, iNodeEnter, iNodeExpand, and iNodeColor.

```
; $construct method of remote form; contains iTreeList list var
Do iTreeList.$define(iNodeRoot, iNodeChild, iNodeIcon, iNodeIdent,
    iNodeEnter, iNodeExpand, iNodeColor)
Do iTreeList.$add("Icons", "", 1651+k16x16, 1, 0, 0, 0)
Do iTreeList.$add("Icons", "1650", 0, 1650, 0, 0, 0)
Do iTreeList.$add("Icons", "1652", 0, 1652, 0, 0, 0)
Do iTreeList.$add("More Icons", "", 1652+k16x16, 2, 0, 0, 0)
Do iTreeList.$add("More Icons", "1662", 0, 1662, 0, 0, 0)
Do iTreeList.$add("More Icons", "1660", 0, 1660, 0, 0, 0)
```

The tree list reports a number of events in response to clicks on nodes or icons. All the events pass the `pNodeIdent` event parameter containing the ident of the node or icon clicked. Note that you need to enable these events in the `$events` property for the tree list for them to be triggered.

```
On evWTreeNodeClick
    ; node is clicked, pNodeIdent contains the ident of node
On evWTreeNodeDClick
    ; node is double clicked, pNodeIdent contains the ident of node
On evWTreeIconClick
    ; an icon is clicked, pNodeIdent contains the id of the icon
On evWTreeNodeCollapse
    ; node is collapsed
On evWTreeNodeExpand
    ; node is expanded
```

Wash

The Wash component is contained in the FORMWASH component file, and is located under WEB Background Objects group in the Component Store.

The Wash form component lets you add a color fade effect to the area of the form that it occupies. You can specify the start and end color of the wash of the object, as well as the wash direction.

The wash object is a background component which means when you use the notation you have to reference the object using its ident, for example, the following notation assigns a start and end color and a wash direction to the component which has the ident 1034.

```
Do $cinst.$bobj.1034.$startcolor.$assign(kGreen)
Do $cinst.$bobj.1034.$endcolor.$assign(kMagenta)
Do $cinst.$bobj.1034.$direction.$assign(kWASHleft)
```

Chapter 5—Remote Tasks

A *remote task* is a type of Omnis class that handles the connection between a remote form on the client and your Omnis application, and in some cases performs some server-side event handling and processing. Therefore, all remote forms require a remote task. When you create a remote form using one of the form wizards, Omnis creates a remote task class for you automatically, so in most cases you can use this one. If you want to create your own remote task classes, the Component Store contains a number of templates and wizards.

As an alternative to using the Omnis web client and remote forms, Omnis lets you interact with your Omnis application and database over the Internet using standard html forms and remote tasks. In this case, your html forms connect to a remote task class directly, and no remote forms are required for this type of interaction. See a later section in this chapter for more details about using standard html forms for direct access.

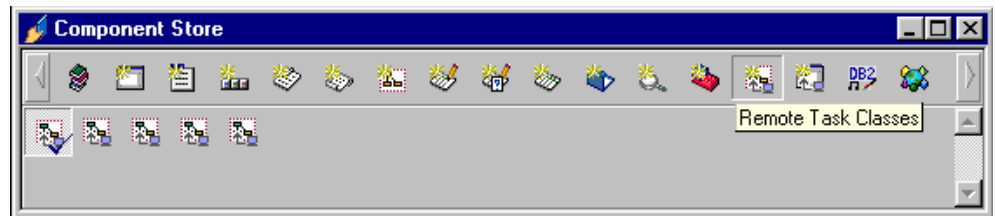
Creating Remote Task Classes using Wizards

You can create an Omnis remote task from scratch or using one of the templates or wizards provided in the Component Store. The following templates are available:

- ☐ **Plain Remote Task**
creates an empty remote task containing an \$event method with code for evBusy and evIdle events
- ☐ **Monitor**
creates a task and window to monitor remote connections.
- ☐ **HTML Report**
creates a task to generate html reports on the fly
- ☐ **Submit**
creates a task and html file containing a submit form which interacts directly with Omnis.

To create a remote task class using a wizard

- Open your library in the Omnis Class Browser
- Display the classes in your library using the View>>Down One Level menu option on the Browser menu bar
- Press F3/Cmnd-3 to open the Component Store or bring it to the top
- Click on the **Remote Task Classes** button to show the remote task wizards and templates



- Drag the wizard onto your library in the Class Browser
- Name the new class and press Return
- Follow the instructions in the wizard

Plain Remote Task Wizard

The Plain Task wizard creates a basic template remote task that is suitable for linking to most simple remote forms. The Plain remote task also has an `$event()` method containing a template event handling method that detects `evBusy` and `evIdle` events in the task. You can add your own code to handle these events.

The Plain remote task has a `$construct()` method containing a parameter variable called *pParams* of type *Row Variable*. This row variable can receive all the parameters of the ActiveX/plugin, including the remote form name, task name, etc, and up to nine additional parameters which are embedded in the html page containing the form.

When you create a task using the Plain Task wizard you can specify the **Inherit from Monitor task** option. This option adds a set of “monitor” classes to your library which allows you to record client connections associated with the new plain task you are adding to your library. If you check the Monitor option, the wizard prompts you for details about the new monitor task. If your library does not contain a monitor task, you need to specify the **Create New Monitor Task** option. If, however, your library contains a monitor task, you can specify the **Use Existing Monitor Task** option to add the new plain task you are

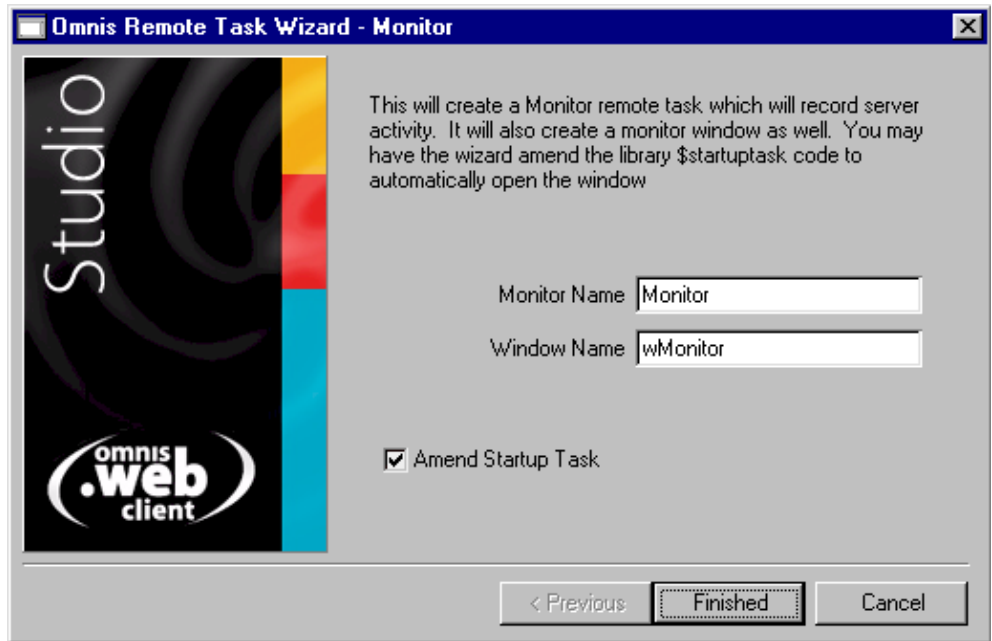
currently adding to your library to the existing monitor. See the next section about the *Monitor Wizard* for further details.

Monitor Remote Task Wizard

The Monitor wizard creates a number of “monitor” classes, including a new task and monitor window, that allow you to record remote connections between clients using the Omnis web client and the Omnis Server.

To create a Monitor window and associated classes

- Display the classes in your library using the View>>Down One Level menu option on the Class Browser menu bar
- Press F3/Cmnd-3 to open the Component Store or bring it to the top
- Scroll the top toolbar in the Component Store and click on the **Remote Task Classes** button to show the remote task wizards and templates
- Drag the **Monitor** wizard onto your library in the Class Browser
- Name the new class, or keep the default name, and press Return
- Follow the instructions in the wizard



The wizard prompts you to enter the name of the new Monitor remote task and window, in most cases however, you can accept the default names. The **Amend Startup Task** option lets you add code to the Startup_Task in the current library to open the Monitor window at startup; this is checked by default, and in most cases you should let Omnis change your startup code.

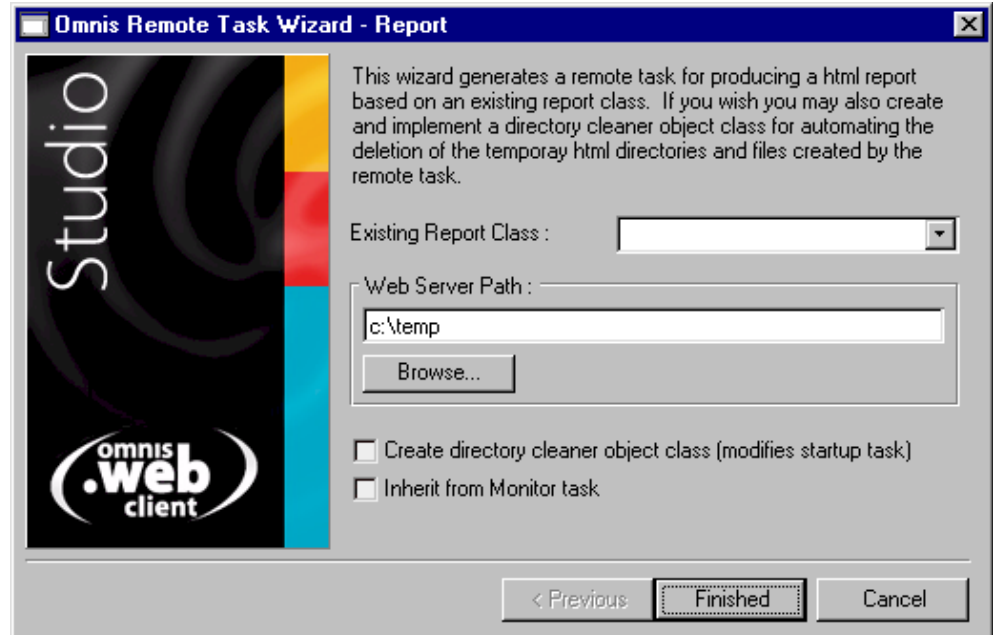
Monitor Window

The Monitor window, called wMonitor by default, has three panes. The **Connections** pane shows the connections grouped by remote form name. The **History** and **Server Usage** panes let you monitor the traffic flow on your Omnis Server and provide some general information about server usage. You can print the server usage using the **Print Report** button.

HTML Report Task Wizard

The HTML report task wizard creates a remote task that generates an Html report on-the-fly and returns it to the client. The \$construct() method contains code to print a report to html using the temp directory object, and return an URL or error message to the client.

The wizard lets you base your Html report on an existing Omnis report class. You can also specify the folder where the temporary report pages are placed, and you can add an object class to your library to clean out the contents of the temporary folder. In addition, you can add the new HTML report task to the Monitor task.



The HTML Report wizard creates an html file containing a Print Report button, using the name TaskName.htm, and places it in the Omnis/HTML folder. When the button on the html form is pressed, the \$construct() of the task creates a new html page in the webserver\omnishtml\[reportname] folder which contains the report. This page is then returned to the user. The Cleaner object sweeps the omnishtml folder in your webserver and deletes expired reports every minute.

Submit Remote Task Wizard

The Submit wizard creates a remote task and html file containing a submit form. The html form created by this wizard allows direct access to your database, and does not use a remote form or the Omnis web client. See a later section in this chapter for more details about using standard html forms for direct access.

The \$construct() method in the Submit task contains variables to handle the data from the form in the associated html file. The wizard places the Submit html file in your Omnis/HTML folder. The Submit task can be inherited from the monitor task.

The first pane in the wizard lets you define the fields to appear in your html Submit form. A Submit and Reset button are included on the form automatically.

Control Type	Name
--------------	------

Properties

Control Type: Text Field

Name: TextField1

Label: TextField

☒ Inherit from Monitor task

< Previous Next > Cancel

You will need to amend the \$construct() code of the remote task to return the URL you want to return when the user has pressed the Submit button on the form. This is documented in the task's \$construct() method.

Creating a New Remote Task Class

The Remote Task wizards in the Component Store let you create many types of remote task, but you can create a completely empty task using the New Remote Task template either from the Component Store or using the Class>>New option in the Class Browser.

To create a new remote task class

- Open your library in the Omnis Class 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 Remote Task” from the Component Store onto the Browser

or

- Select Class>>New>>Remote Task from the Browser menu bar
- Name the new task
- Double-click on the task class to modify it

You modify a remote task class in the method editor. You can place in the \$construct() method any code that you want to run when the remote task is instantiated. You can add any other custom properties and methods to the task, as well as any type of variable including instance variables.

Remote Task Instances

When the Omnis web client first connects, Omnis first creates an instance of the Remote Task Class as specified by the Remote form class to which you are connecting (\$designtaskname property of the form). The \$construct() method of the Remote Task Class can specify a parameter variable of type Row as its first parameter, containing up to 9 optional parameters that you can specify in the html definition of the web client plug-in. If you want to check the value of the Param1 parameter you can use *params.pOption1* assuming you named the row variable *params*.

When working with remote forms and the remote task \$construct() method terminates, it should do so without returning a value if the connection is to go ahead, that is, *Quit method* without specifying a return value. If you need to pass an error back to the client, you can use *Quit method 'The Error message'*. In this case the task instance is destroyed and the connection is terminated. If you don't return an error, Omnis will now instantiate an instance of the remote form class and call the \$construct method of the form.

Events

For remote tasks, the `evBusy` and `evIdle` events are sent to the `$event()` method during the lifetime of a connection: `evBusy` is sent when Omnis receives a request from a client, `evIdle` is sent when Omnis is about to return the result of a request, i.e. the task instance is about to become idle. The following example, shows the code for the `$event()` method in the Monitor task created using the Monitor task wizard:

```
On evBusy
  If iMonitorOpen
    Do iMonitorRef.$setstatus($cinst,kTrue) Returns lServerBusyFlag
    If lServerBusyFlag
      Quit event handler (Discard event)
      ; server cannot handle request
    End If
  End If
On evIdle
  If iMonitorOpen
    Do iMonitorRef.$setstatus($cinst,kFalse)
  End If
On evRejected
  Do $cinst.$showmessage(pErrorText)
```

In addition, tasks report the `evRejected` event which is generated when Omnis rejects a connection by a client. Usually this occurs if there are too many users trying to connect to Omnis, or `$maxusers` of the remote task has been exceeded. The parameter `pErrorText` is "Too many users connecting to server" for the first case, and "Too many users connecting to task [taskname]" for the second.

Tasks report the `evPost` event when Omnis receives a request from a standard HTML form. This only applies when a remote task instance originally constructed for an HTML form request, returns `kFalse` from `$anclose`. Subsequent requests from a form which identifies the connection id of the remote task arrive as `evPost` events. The row variable parameter `pPostData` contains the form parameters from the client, and you use the field reference parameter `pPostResult` to return the result to the client.

Client Access Properties

Remote tasks instances have a number of properties that let you monitor connections to your Omnis application.

- ❑ **`$connectbytessent`**
specifies the number of bytes which have been sent to the client during the connection. This property is set after `$construct()` has been executed.

- ❑ **\$requests**
specifies the number of events executed on the server. Excludes connect and disconnect messages. Updated prior to evBusy message.
- ❑ **\$reqtothbytesreceived**
the total number of bytes received from the client for all requests. To calculate an average per request, you can divide this value by \$requests. Updated prior to evBusy message.
- ❑ **\$reqtothbytessent**
the total number of bytes sent to the client for all requests. To calculate an average per request, you can divide this value by \$requests. Updated prior to evIdle message.
- ❑ **\$reqmaxbytesreceived**
The largest block in bytes received from the client for all requests. Updated prior to evBusy message.
- ❑ **\$reqmaxbytessent**
The largest block in bytes sent to the client for all requests. Updates prior to evIdle message.
- ❑ **\$reqcurbytesreceived**
The number of bytes received from the client for the current request. Updated prior to evBusy message for the current request.
- ❑ **\$reqcurbytessent**
The number of bytes sent to the client for the current request. Updated prior to evIdle message.

Timeouts

You can control how long someone is connected to the Omnis Server and how long a single client connection can remain idle, using the following properties.

- ❑ **\$maxtime**
the maximum time in minutes that a client is allowed to stay connected; the default value is 0 which means the client can stay connected indefinitely.
- ❑ **\$timeout**
the maximum time in minutes that a client is allowed to stay idle; the default value is 0 which means the client is allowed to stay idle indefinitely.

Client Connections

Remote task instances have some properties that tell you about the current client connection.

- ❑ **\$clientaddress**
the TCP/IP address of the current client.

- ❑ **\$connectionid**
the id of the current client connection; ids are allocated dynamically by the Omnis Server and numbers are not reused unless the server is restarted.
- ❑ **\$connectiontime**
the time and date the client connected to the Omnis Server, i.e. the time the current task instance was instantiated.
- ❑ **\$lastresponse**
the time and date the client last accessed the current task instance on the Omnis Server.

Secure Sockets

Omnis will use secure sockets mode (https), if available, if the `WebServerUrl` parameter in your html specifies a full URL using the https: prefix. However, remote tasks have the `$issecure` property that lets you turn secure mode on and off dynamically, by assigning to the property for the current task at runtime.

Chapter 6—Using Standard HTML Forms

In addition to the Omnis web client, Omnis Studio lets you interact with your Omnis application using standard html forms; this approach is sometimes referred to as ‘ultra-thin’ since the client’s browser does not require a plug-in. In this case, your html form has to connect to an Omnis remote task class and does not use a remote form. For example:

```
<form method="GET" action="/cgi-bin/omnisapi.dll">
<input type="hidden" name="OmnisServer" value="PortNumber">
<input type="hidden" name="OmnisClass" value="RemoteTaskName">
  <input type="hidden" name="OmnisLibrary" value="LibraryName">
<p><input type="password" name="pPassword" size="20"></p>
  <p><input type="text" name="pQuery" size="80"></p>
  <p><input type="submit" value="Submit" name="B1">
  <input type="reset" value="Reset" name="B2"></p>
</form>
```

The form has the special fields:

- ❑ **OmnisServer**
specifies the port number or service name of the Omnis runtime, that is, the value specified in the \$serverport preference in the Omnis runtime.
- ❑ **OmnisClass**
the name of the Remote Task Class to connect to.
- ❑ **OmnisLibrary**
the internal name of the library to connect to, that is, the name of your library less the .lbs extension; in this case, the library and the Omnis runtime must be running for the form to connect. If the Omnis Server is not on the same machine as your web server,

The following example html source code implements a feedback form. The Omnis specific parameters are marked in bold; the remainder of the source specifies the form fields and text labels in the form, including a standard Submit button.

```
<form method="GET" action="/cgi-bin/omnisapi.dll">
  <input type="hidden" name="OmnisClass" value="tThinClientFeedback"> ;;
  the remote task name
  <input type="hidden" name="OmnisLibrary" value="Webclient"> ;; the
  library name
  <input type="hidden" name="OmnisServer" value="5912"> ;; the port number
  <table border="0" cellspacing="0" cellpadding="0" width="760">
    <tr>
      <td width="788" valign="top"><div align="right">
        <p><strong><font face="Arial">Developer
          Name:</font></strong></td>
        <td width="564" height="25"><font face="Arial">
          <input type="text" name="Name" size="27"></font></td>
      </tr>
      <tr>
        <td width="788"><div align="right">
          <p><strong><font face="Arial">Serial
            No:</font></strong></td>
          <td width="564" height="25"><font face="Arial">
            <input type="text" name="Serial" size="27"></font></td>
        </tr>
        <tr>
          <td width="788" valign="top"><strong><font face="Arial">
            <div align="right">
              <p>Platform:</font></strong></td>
          <td width="564" height="23"><table border="0"
            cellspacing="0" cellpadding="0" width="520">
              <tr>
                <td width="135"><font face="Arial">
                  <input type="checkbox" name="Macintosh" value="YES">
                  <strong>Macintosh</strong></font></td>
                <td width="385"><font face="Arial">
                  <strong><input type="checkbox" name="Windows"
                    value="YES">Windows</strong></font></td>
              </tr>
            </table>
          </td>
        </tr>
        <tr>
          <td width="788" valign="top"><strong><font face="Arial">
            <div align="right"><p>Client:</font></strong></td>
          <td width="564" height="23">
            <table border="0" cellspacing="0" cellpadding="0"
              width="601">
```

```

        <tr>
            <td width="136" height="13"><font face="Arial">
                <input type="checkbox" name="ActiveX" value="YES">
                <strong>ActiveX&nbsp;&nbsp;&nbsp;</strong></font></td>
            <td width="135" height="13"><font face="Arial">
                <strong><input type="checkbox" name="Netscape"
                    value="YES">Netscape </strong></font></td>
            <td width="330" height="13"><font face="Arial"><strong>
                <input type="checkbox" name="RawHTML"
                    value="YES">RawHTML</strong></font></td>
        </tr>
    </table>
</td>
</tr>
<tr>
    <td width="788" valign="top"><strong><font face="Arial">
        <div align="right">
            <p>Comments:</font></strong></td>
    <td width="564" height="249" valign="top">
        <font face="Arial"><textarea rows="11" name="Comments"
            cols="57"></textarea></font></td>
    </tr>
</table>
<div align="center"><center><p><font face="Arial">
    <input type="submit" value="Send Comments" name="B1"></font></p>
</center></div>
</form>

```

When the ‘Submit’ button is pressed, the web server plug-in (e.g. omnisapi.dll) is executed and passed all the form’s parameters. The web server plug-in sends the request to the Omnis queue. Omnis creates an instance of the specified Remote Task Class and calls its \$construct method. When the \$construct terminates, the task instance is destroyed.

Task Wizards and \$construct

All the remote tasks created using the remote task wizards contain a \$construct() method containing a parameter variable of type Row and called pParams. All html form fields in the form can be accessed from this row variable. For example, using the form above, you can check the value of the password field using *pParams.pPassword*. Note that the column name in the row variable is the name given to the html web component.

When the \$construct method of the task terminates, it can return the URL of the result page if appropriate, that is, *Quit method* ‘www.mywebsite.com/result.htm’. The result could be a simple Thank You page telling the client that their details were received, or it could be an html report page generated in Omnis using the html printing device.

If you need to return an error you can use *Quit method* ‘nnn The Error message’, where nnn is a three digit error code followed by a single space before the error text.

In addition to being able to return an URL and an error message to the client browser, it is possible to stream back html to the client, i.e. the html that is to be returned to the client is contained in a variable. This variable is sent back to the client, for example:

```
Quit method {iHtmlVar}
```

Chapter 7—Multi-Threaded Server

The Omnis Server allows multiple requests to be processed allowing smoother allocation of available processor time and avoiding any lengthy delays on the client. To handle these multiple requests, the Omnis Server is multi-threaded. By default, the Omnis Server is a standard single-threaded server handling client requests in a strictly first-come, first-serve basis; client requests are queued, with each request being handled only when the previous request has completed. You can however handle multiple client requests concurrently using the Omnis Multi-threaded Server, which you can enable by executing the *Start server* command on the Omnis server.

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

Multiple Method Stacks

The standard single-threaded Omnis Server has only a single server stack to process methods. Broadly speaking, once a method call has been pushed onto the method stack no other method call can begin to execute until the first call has completed. For the majority of web client applications this is fine for processing events, particularly if some processing is performed on the client and your web server receives relatively few hits or requests for data. By contrast, the Omnis Multi-threaded Server contains a pool of method stacks which are available to process multiple client requests, and this is appropriate for more data intensive web applications where multiple calls to a server database are required, or for websites that receive higher volumes of traffic. When a request to execute a method is received from a web client, that method call is pushed onto any unused stack or, if there are no unused stacks, the message is queued until one becomes available. Each method stack runs in its own thread, which means that if a method stack is stalled (for example, it is waiting for the database server) the other stacks will continue to execute.

Potentially, the Multi-threaded Server may have to cope with a very large number of simultaneous clients, each with their own remote form and remote task instances. Typically though, a small proportion of clients will require the use of the server at any one time. In fact, multi-threading does not increase the server processor time available, it just allows the available processor time to be allocated in a smoother way. The method stack pool mechanism allows a balance to be struck between performance and server resources - the

number of method stacks in the pool is configurable with the `$root.$prefs.$serverstacks` property, which is set to 5 by default.

As method stacks are allocated dynamically, it is very likely that a remote client will not get the same method stack every time it executes a method on the server. Each method stack contains its own state which, apart from during an individual method call, does not belong to any particular client. This state includes the Current Record Buffers (CRBs) for all files and variables (apart from class variables) and such modes as the current list. A client cannot rely on any properties or values of this state being preserved across different method calls, the only thing belonging to the client are its instance and task variables. So a client must do such things as setting the main file and current list each time one of its methods is executed, and should not rely on such things as the values of memory-only fields being maintained across method calls. As a special case, the class variables for the remote task and remote form classes are shared amongst all clients so can be used to hold shared data (see below for the warnings about the care needed when using shared variables).

Using the Server

When the Multi-threaded Server starts up, it opens the libraries, datafiles and SQL session pools required by the clients (see below for description of a SQL session pool). You need to issue the *Start server* command to cause the method stacks and associated threads to be created, whereupon Omnis starts to listen for client requests. The *Start server* command can specify an optional stack initialization method; when specified, this method is pushed onto every client method stack and allowed to execute (so if `$serverstacks` is 5 it will execute five times), so it can be used to initialize the state of the method stacks. The *Start server* command generates a fatal error if, due to lack of resources or some other reason, it is unable to complete successfully.

When you want to stop the server, you should issue the *Stop server* command, but quitting the Studio program achieves the same result.

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

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

Database Access

If you are accessing a server database in your web application, and using the Multi-threaded Server, you must use the Data Access Modules (DAMs) introduced in Omnis Studio 3. The new DAMs and DAM API in Studio 3 supports multi-threading on the Omnis Server. Using the DAMs, you can connect to Oracle (7 & 8), DB2, Sybase, and many others via ODBC. The old DAMs supplied with Studio v2 or earlier DO NOT work on the Multi-threaded Server. In addition, any methods or code that uses the old DAMs or the old DAM interface will not work on the Multi-threaded Server.

The new DAMs are implemented as non-visual external components. The new DAMs will not work with the current 4GL SQL commands, rather you need to use object variables based on the new non-visual DAMs, and interact with a DAM using the methods of the object. Using the non-visual approach, you create object variables of a particular DAM class which are instantiated when the object is created by the component. There is a group of common methods which apply to all DAM objects and a set of DAM specific methods based on the type of object. Please see the *Omnis Programming* manual for full details about accessing your server database using the Omnis DAMs.

Omnis datafile access

You can access the built-in Omnis database (i.e. Omnis datafile) on the Multi-threaded Server. The list of open datafiles is shared amongst all the method stacks and all the datafiles which might be used by the clients should be opened before the *Start server* command is issued.

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

Omnis Server Commands

There are some new commands that you can use to control the Multi-threaded Server which are contained in the Threads... command group in the Omnis method editor.

Start server

The *Start server* command is used to create the client method stacks and associated threads and start the thread which listens for web client requests. It takes an optional stack initialization method as a parameter. The flag is cleared if used in a single threaded Omnis

server or your serial number does not allow clients to connect. A fatal error is generated if for some other reason it is not possible to create the stacks and threads and start the listener.

Stop server

The *Stop server* command stops the server from responding to client requests. Once the server has been started you should stop it before quitting Omnis, before using Omnis for anything apart from serving client requests (e.g. running a standard LAN-based Omnis application), or before opening or closing any Omnis datafiles or libraries.

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

Begin and End critical block

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

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

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

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

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

Yield to other threads

The *Yield to other threads* command is a hint that the executing thread is waiting for other threads and is prepared to yield its processor time. It can be used when waiting for semaphores (since with the Multi-threaded Server another client stack could be holding the semaphore), as follows:

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

Commands which are not available to a client

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

```
Enter data
Prompted find
The Libraries... group
The Classes... group
Pre V30 SQL commands...
Open data file
Prompt for data file
Create data file
Close data file
Open lookup file
Close lookup file
The Data management... group
The Message boxes... group
The Debugger... group
Quit Omnis
```

Any other command which would cause a dialog to be displayed on the server is not available for methods running on the Multi-threaded Server. In addition, there's a lot of notation, such as the notation for opening and closing libraries and datafiles, that will not work in a method running on the Multi-threaded Server.

Chapter 8—Server Load Sharing

Load sharing allows a pool of multi-threaded Omnis Server processes, running on one or more machines, to serve clients. Once a client connects to an Omnis server process, all subsequent requests for that client need to be handled by the same Omnis server process. Therefore, load sharing provides a mechanism that assigns a new client connection to an Omnis server process. Load sharing is implemented for Win32 and Linux Omnis servers only.

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

When a new connection arrives at the Web Server plug-in (ISAPI, CGI or Apache module located in the cgi-bin or scripts folder), the plug-in recognizes the new syntax of the OmnisServer parameter. If it is prefixed by a pool name, the plug-in connects to the load sharing process, and sends it a message that asks for the address of a server process in the pool. The load sharing process typically returns the address and port number of the least busy process in the pool. The plug-in then connects to this process, and sends the web client connection to it. When the plug-in responds to the client, it includes the address of the Omnis server process in the response.

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

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

You can configure the time interval between polls of each Omnis server process via the .ini file. Once every 10 or 20 seconds is usually frequent enough.

Enabling Load Sharing

To enable the load sharing process you need place the LSP program on your web server. This is a single executable called `Omnislsp.exe`. A configuration file (`omnislsp.ini`) must be made to accompany the `Omnislsp.exe` program, this takes the following format:

```
[Setup]
Port=6001
QuietMode=0
BucketSize=100
LogLineThreshold=16

Pool1=Omnis
[Omnis]
PollTimer=10
Server1=123.145.71.123:7001
Server2=123.145.71.124:7002
```

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

(With the `omnislsp.ini` in the same directory)

You also need to edit the 'OmnisServer' parameter in your html file containing the web client plug-in, for example:

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

where `Omnis` is the name of a pool of `Omnis` server processes and `6001` is its port number.

On the LSP servers

The servers specified in the `omnislsp.ini` do not run any `www` server software. The servers may be stopped and restarted without the need to stop the LSP.

Load Sharing Mechanism

The load sharing process periodically polls the processes in a pool of `Omnis Server` processes. Each server returns the current number of connections to the server, the maximum number of concurrent connections allowed to the server (specified by the serial number), the number of current requests, and the total elapsed time in milliseconds taken to process the requests. The load sharing process organizes the servers into groups, based on the results of the information returned from polling the servers.

When a connection request arrives at the load sharing process, it allocates a server to the request as follows. It traverses the groups, looking for a server that has some free

connections. Within a group, it looks for the server with the smallest percentage of connections in use, using the results of the last poll. If there is more than one server with the same smallest percentage of connections in use, the process allocates the connection to the server to which it least recently allocated a connection. At this point, the load sharing process also updates the connection statistics from the last poll, to reflect the new connection. The traversal stops when a free process has been found.

Using version 2.x Web Clients

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

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

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

Chapter 9—Deploying Your Omnis Web Application

Having designed your remote forms and tasks in your Omnis library, you are ready to deploy your application to the web. You need to place your Omnis library on a Win32 or Linux server with the Omnis runtime, and upload your html pages to your intranet or internet web server. When users access your website for the first time, they need to download the Omnis web client. You can use the installers supplied on the Omnis Studio CD or create your own installers, but in all cases, users need to download the web client before using your web application.

This chapter describes how you:

- ☐ set up your html pages containing the Omnis web client;
this involves editing the template html pages created for you by the form wizards
- ☐ install your web server and the web server plug-in supplied by Omnis
- ☐ install the Omnis Server, i.e. the Omnis runtime and your Omnis library
- ☐ copy your html files and web client installers to your web server;
you may need to create your own installer for users to download from your website

Editing your Html Pages

The Omnis web client is placed inside an html page using the <OBJECT> or <EMBED> tag. You can edit the parameters for the web client plug-in using any html editor or a standard text editor. When you press Ctrl/Cmnd-T to test your remote form in design mode, Omnis creates a template html file automatically, and places it in the **html** folder under the main Omnis folder. You can use these template html files as a basis for your html pages or copy the embedded plug-in and its parameters to your own html pages. Note that the templates contain the html code for the ActiveX for IE and the Netscape plug-in, so if you are using IE only you can comment out the Netscape plug-in from the template html page. Netscape on the other hand ignores the ActiveX code. Alternatively, you can use Javascript, server side includes (SSIs), or an Active Server Page (ASP) to detect the type of browser on the client machine and display the appropriate web client plug-in, either the ActiveX or Netscape plug-in.

Embedding the ActiveX Web Client

The html code and parameters for the ActiveX are as follows.

```
<object classid=" clsid:13510606-30FA-11D2-B383-444553540000"
    width="500" height="250">
<param name="OmnisServer" value="ServerPortNumber">
<param name="OmnisLibrary" value="LibraryName">
<param name="OmnisClass" value="RemoteFormName">
<param name="WebServerUrl" value="www.yourwebserver.com">
<param name="WebServerScript" value="/cgi-bin/omnisapi.dll">
<param name="Param1" value="pOption1=Data">
...
<param name="Param9" value="pOption9=Data">
</object>
```

The Omnis web client ActiveX has the following parameters.

- ❑ **classid**
the id of the ActiveX object; this includes the height and width of the object which ought to be the same as the remote form; the plug-in size does not change if you change the size of your remote form, so you may need to edit these values.

- ❑ **OmnisServer**
the port number of the server, or service name, which is registered on the server, and specified in the \$serverport preference in Omnis. The OmnisServer parameter can also be in the form `IpAddress:port`, if the Omnis Server is running on a different machine to your web server. For example,
`"111.222.333.444:5912"`
`"111.222.333.444:omnis"` ;; where 'omnis' is a service name
`"www.myhost.com:5912"`
`"www.myhost.com:omnis"`
- ❑ **OmnisLibrary**
the internal name of the library to connect to; the default is the library file name minus the .lbs extension.
- ❑ **OmnisClass**
the name of the remote form to connect to.
- ❑ **WebServerUrl**
the URL of your web server; the URL can be of the form:
`IP_Address:PortNumber` or
`DomainName:PortNumber`
 If a port number is not specified, port 80 will be used when communicating with the web server.
 For local testing the WebServerURL can be set to 127.0.0.1 (i.e. the loopback address) or it can be the IP address of your machine.
- ❑ **WebServerScript**
the location of the web server plug-in, such as the `omnisapi.dll`, on your web server; typically the `/cgi-bin` folder.
- ❑ **Param1 to Param9**
specifies additional parameters, which can be sent to the Omnis Server. In the value clause you have to specify the parameter name and the data separated by an equal sign.

The following html code allows a connection to a remote form called **rfBooks** in the Books example on a web server in Mitford House, Omnis' main development center.

```

<object classid="clsid:13510606-30FA-11D2-B383-444553540000"
    width="298" height="287">
    <param name="_Version" value="65536">
    <param name="_ExtentX" value="7161">
    <param name="_ExtentY" value="7373">
    <param name="_StockProps" value="0">
    <param name="OmnisServer" value="5912">    ;; the port number
    <param name="OmnisLibrary" value="WEBCLIENT">    ;; the library name
    <param name="OmnisClass" value="rfBooks">    ;; the remmote form
    <param name="WebServerUrl" value="mhthinserver.mh.omnis-
        software.com">    ;; the web server name
    <param name="WebServerScript" value="/cgi-bin/omnisapi.dll">
        ;; the location of the web server plug-in omnisapi.dll
</object>

```

You can examine the template html pages in the Omnis\HTML folder to see how the Omnis web client is embedded. The template pages have the port number/name, library name, and remote form class parameters filled in, but you will need to change or add the other parameters, such as WebServerUrl and WebServerScript, to allow remote access to the remote form.

Embedding the Netscape Plug-in Web Client

The html code and parameters for the Netscape plug-in are as follows.

```

<EMBED type=application/Omnis-RCC-plug-in
    name="rccl"
    width=500
    height=500
    OmnisServer="ServerPortNumber"
    OmnisLibrary="LibraryName"
    OmnisClass="RemoteFormName"
    WebServerUrl="www.yourdomainname.com"
    WebServerScript="/cgi-bin/omnisapi.dll">

```

- ☐ **OmnisServer**
the port number of the server, or service name, which is registered on the server, and specified in the \$serverport preference in Omnis.
- ☐ **OmnisLibrary**
the internal name of the library to connect to; the default is the library file name minus the .lbs extension.
- ☐ **OmnisClass**
the name of the remote form class to connect to.
- ☐ **WebServerUrl**
the URL of your web server DLL.

❑ **WebServerScript**

the location of the web server plug-in, such as omnisapi.dll, on your web server; typically the /cgi-bin folder.

Autodetecting Browser/Platform

You can use Javascript to detect the client's browser and platform type and display the appropriate web client plug-in. Using Javascript you can either detect the browser type and display one of two separate html files, or you can display the appropriate plug-in in a single html file. The following example index.html would be at the root of your webserver. The Javascript detects the user's platform and redirects the browser to either the 'msindex.html' or the 'netscapeindex.html' file which contain the ActiveX and Netscape plug-in respectively. You could use the same script to detect the client's platform and display the appropriate web client installation page containing either the ActiveX download or Netscape plug-in installer.

```
<SCRIPT language="JavaScript">
<--
    if (navigator.appName.indexOf('Microsoft') != -1)
        window.location.href = 'msindex.html';
    else
        window.location.href = 'netscapeindex.html';
//-->
</SCRIPT>
```

Alternatively, you can use Javascript embedded in a single html page to autodetect the browser/platform type of the client and display the appropriate plug-in in your html page. The following example Javascript detects if the client's browser is Internet Explorer under 32bit Windows and if so the ActiveX is displayed, otherwise the Netscape plug-in is displayed in the page.

```
<SCRIPT language="JavaScript">
    if(navigator.appName == "Microsoft Internet Explorer" &&
        navigator.platform == "Win32") {
        document.write("<center><object
            classid='clsid:13510606-30FA-11D2-B383-444553540000'
            width='746' height='585'>");
        document.write("<param name='_Version' value='65536'>");
        document.write("<param name='_ExtentX' value='7161'>");
        document.write("<param name='_ExtentY' value='7373'>");
        document.write("<param name='_StockProps' value='0'>");
        document.write("<param name='OmnisServer' value='6182'>");
        document.write("<param name='OmnisLibrary' value='LIBNAME'>");
        document.write("<param name='OmnisClass' value='rfMyForm'>");
        document.write("<param name='WebServerURL'
            value='portal.mycompany.net'>");
    }
```

```

        document.write("<param name='WebServerScript'
            value='/cgi-bin/omnisapi.dll'>");
    document.write("</object></center>");
} else {
    document.write("<center><embed
        type='application/Omnis-RCC-plug-in'
        name='rccl' width='746' height='585'
        OmnisServer='6182' OmnisLibrary='LIBNAME'
        OmnisClass='rfMyForm'
        WebServerUrl='portal.mycompany.net'
        WebServerScript='/cgi-bin/omnisapi.dll'></center>");
}
</SCRIPT>

```

CGI

If your web server does not support the Microsoft ISAPI interface you will have to use the `nph-omniscgi.exe` cgi program, or the Apache module under Linux. In this case, you need to add the `nph-omniscgi.exe` program or the Apache module to your `cgi-bin` folder and edit the **WebServerScript** parameter in your HTML file(s).

Omnis Server Configuration

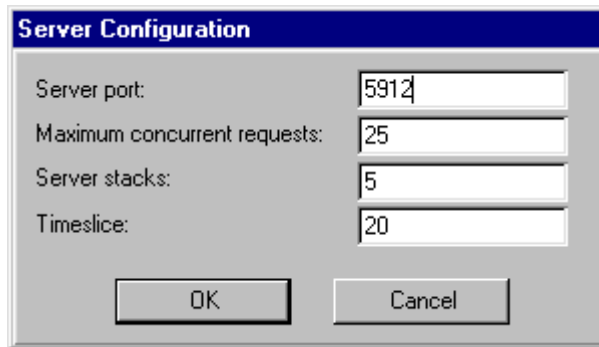
The Omnis Server comprises the Omnis runtime engine and your Omnis library. This section deals specifically with the configuration of the Omnis Server, installed from the Omnis Server folder on the Studio CD. You need to install the Omnis runtime on a Win32 or Linux server and serialize it with a Web serial number. Note that you cannot use a standard multi-user runtime serial number for your Omnis server, in this case you must use a web serial number designed for Omnis web client access. The server does not need to be located on the same machine as your web server, but it must be a Win32 or Linux server. For testing and debugging you can use the Omnis development version, but for deployment you should use an Omnis runtime executable.

Setting the Omnis Port Number

You need to set the port number or service name of the Omnis runtime server. The port number must be available on the Omnis Server. The number can be between 1 and 32767, but you must not use 80 or any other number used for e-mail, FTP, or other services. You should use a high number, preferably a four or more digit number, for example, 5912. You can set the Omnis server port in design mode, in the Omnis Server Configuration dialog, or using the command *Calculate \$prefs.\$serverport as 5912*.

The Omnis server port number is stored in the `Omnis.CFG` file in the `STUDIO` folder. Under Windows NT, there is a file called 'Services' in the folder `WINNT/System32/drivers/..` which lists the ports already registered by other programs.

In the development version of Omnis, you can set the port number in the Omnis preferences via the **Tools>>Options/Preferences** menu option. This option opens the Property Manager showing the Omnis preferences. You can set the Omnis server port in the \$serverport property. However, since you'll be running your Omnis web application using the runtime version of Omnis, you have to set its server port number too. You can set the port number in the Omnis server via the Server Configuration dialog, available under the **File>>Server Configuration** menu option (this item is visible in Win32 or Linux versions only, and only if the runtime is serialized with a web serial number).



The Server Configuration dialog lets you enter the server port and server maximum concurrent requests preferences; the latter is specified by the serial number of your Omnis Server, but if required you can further limit the number of connections by decreasing this number. The Server stacks (\$serverstacks) parameter lets you specify the number of method stacks on the Omnis Server. The Timeslice (\$timeslice) parameter lets you specify the duration of the execution time slice for a server thread.

When you click OK in the Server Configuration dialog and if the Omnis Server is already listening, you are prompted to restart Omnis, otherwise the Omnis runtime starts to listen on the specified port using the new parameters.

The web server plug-in (omnisapi.dll) communicates via TCP/IP with Omnis. Therefore your Win32 web server and the Omnis Server both require WinSock. Linux supports TCP/IP automatically.

NT Server Setup

For NT servers only, a program called NTSERV.EXE is provided on the Studio CD, which installs an NT service to run the Omnis server. By default the NT service name is *Omnis Server*. Do not confuse NT service with the TCP/IP service names which you can specify in the \$serverport property for communication. They are unrelated.

If you did not set up Omnis as an NT service during installation you can register it now by running NTSERV.EXE with the following command line:

```
NTSERV -install
```

To remove Omnis as a service, run the following from the command line:

```
NTSERV -remove
```

To run Omnis as console application for debugging, run the following from the command line:

```
NTSERV -debug <params>
```

It is important that whenever you use the NTSERV.exe that the file is in the same folder as the Omnis.exe that you are using.

Web Server Configuration

Installing the Web Server Plug-in

You need to install the web server plug-in Omnisapi.dll, supplied by Omnis, on your web server inside your /cgi-bin folder of your web site. If your web server does not support the Microsoft ISAPI interface you will have to use the nph-omniscgi.exe cgi program, or the Apache module for a Linux web server, and edit your html pages accordingly, for example:

```
WebServerScript="/cgi-bin/omnisapi.dll" ;; for MS ISAPI web servers
```

```
WebServerScript="/cgi-bin/nph-omniscgi.exe" ;; for generic CGI interfaces.
```

Typically a web server will have a place to store all executable code accessible via HTTP over the Internet. This is often the /cgi-bin folder, but it can be any folder configured to allow execution. You need to place the web server plug-in, such as the omnisapi.dll, in this folder and ensure that the web server is set up correctly to enable it to be executed. The name and location of the plug-in is specified in the WebServerScript parameter of your Html files. Omnis sets up this parameter automatically, along with the other plug-in parameters, in the Html pages generated by Ctrl/Cmnd-T (when testing your remote forms in design mode) or using the wizards and templates.

Apache Server Extension

The Apache module for Linux is called mod_omnis.so and is located in the webclient/server folder in the main Omnis tree. The Apache module works with the Red Hat 6.0 version of Apache, and other Linux distributions. This module is faster than the nph-omniscgi server extension.

To install the Apache module

- Copy mod_omnis.so to the Apache modules directory, e.g. on RedHat 6.0 this is /etc/httpd/modules
- Add the following lines at the end of access.conf:

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

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

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

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

ISP Web Hosting

If your website is hosted by a third-party, typically an ISP, they will need to place the Omnis web server plug-in in their cgi-bin folder, and furthermore they need to provide you with a direct connection to the Internet. Your ISP may want to test the web server plug-in, usually the case for any files you place in their cgi-bin folder, and this is often very expensive. Alternatively, you may consider having your own “local” web server specifically for running your Omnis web application and, if necessary, link to it from your main website hosted by your ISP.

If your Omnis web application uses a website hosted by an ISP you will need to adjust your port settings in the Omnis server and html files. In this case you can use DomainName:Port or IPAddress:Port in your port setting.

Secure Sockets

You can use secure sockets (https), if available, if either the page containing the Omnis web client is secure, or your WebServerUrl specifies a full URL using the https: prefix as follows:

<https://remainderOfFullURL>.

In addition, remote tasks have the \$issecure property that lets you turn secure mode on and off dynamically, by assigning to the property at runtime.

Remote Form Components and Installers

To use your Omnis web application, a user must first download and install the Omnis web client. You can provide a page on your website to allow the user to download an installer with a link to each of the installers for each platform and browser type. Alternatively, using automated web client installers, it is possible to allow users to download and fully install the web client automatically when they first visit a web page containing the web client plug-in. These automatic installers are at present available for Windows only, and are described in the *Automatic Installation* section below..

Installers for both the IE and Netscape plug-ins are provided on the Omnis Studio CD, and they are freely available on the Omnis website at <http://www.omnis.net>. If you want to ensure your users get the most up-to-date web client installers, you can link to the Omnis web client download page on our website.

Web Client Components

This section describes the web client engine (or Omnis remote form controller, Orfc) and web components that comprise the Omnis web client. As explained previously, the Omnis web client is an ActiveX or Netscape plug-in. These are small objects which dynamically link in further DLLs, or Shared Libraries under Linux and MacOS. The following components make up the Omnis web client:

- ☐ **Orfc.dll & Orfc.dat**
the ActiveX object (required)
or
np_orfc.dll & np_orfc.dat
for the Netscape plug-in (required)
- ☐ **Orfcgui.dll**
library that provides the web client GUI (required)
- ☐ **Orfcmain.dll**
the web client engine (required)

The remote form class objects are external components written specially for the remote form class. No other Omnis objects can be placed in a remote form class. The standard web client installer includes the standard form fields (FORMFLDS) and background components (FORMBACK) only; the web client engine itself includes the Page pane component. Any other components that are not included in the standard installer (i.e. most components) are downloaded automatically when the user opens a remote form; all the components are stored in a component database on the Omnis Server and are downloaded as required. This means you can provide the standard installer, and rely on the user to download any other

components via the web as required. Alternatively, you may want to create an installer that contains all the components the user is likely to need for your internet or intranet application to avoid any further downloads.

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

When a client connects to the Omnis Server, the server returns a list of available components in the response to the connect request. The web client analyzes the form and the components already installed on the client, and if it needs to, downloads any missing or out-of-date components. The web client prompts the user, asking them if they want to download the new or updated component(s). Assuming the user clicks OK, the web client downloads the component(s), and the remote form is displayed. If the user cancels the download, the form will not be displayed correctly and will be unusable.

Old versions of the web client can safely be used against Omnis Studio 2.2 servers which use the Component Manager. More recent versions of the web client can safely be used against pre-2.2 servers, although of course no automatic download occurs.

Creating your own Web Client Installers

In most cases, you can use the Omnis web client installers provided by Omnis. For example, you can link to the Omnis web client download page to allow users to get the latest version of the web client, or you can place the installers on your website with the appropriate links. Alternatively, you may want to create your own web client installer, either containing a subset of the components we provide, or if you want to add your own web components. You can use any industry-standard Installer package, such as InstallShield, to create your own web client installers.

The basic web client installer contains the Omnis web client files (`Orfc...`) and the basic fields and background objects only (`Formflds` and `Formback`). To create your own web client installer you must use the client version of components, NOT the development ones located in your `Webcomp` folder. The client components are located in the `Omnis/Webclient` folder, then either `IE` or `Netscape` folder for the respective browser type. A complete list of form components and their respective filenames is as follows:

Component file name	Description
Formback	Background Label and Border components
Formcal	Calendar component
Formclock	Clock component
Formfade	Fade picture component
Formfile	File dialog component
Formflds	Field components including Pushbutton, Single line and Multi line entry field, Checkbox, Radio group, List box, Drop list, Combo box, Headed list and Picture
Formgif	Gif component
Formhpic	Hot pic component
Formicon	Icon component
Forminfo	Info or 'cookie' component
Formjpeg	Jpeg component
Formmarq	Marquee component
Formport	Port component
Formpri	Print component
Formprog	Progress component
Formqt3	Quicktime or 'Movie Player' component
Formroll	Roll button component
Formsbar	Side bar component
Formslid	Slider component
Formtbar	Tab bar component
Formtile	Tile component
Formtime	Timer component
Formtran	Transbutton component
Formtree	Tree list component
Formwash	Wash component

Auto downloading the Web Client installer

The following javascript detects the platform and the browser type of the client and downloads the appropriate web client plug-in installer; note with this method the user still has to run the web client installer. The script should be placed in the <HEAD>.. </HEAD>

section of your html page. Note the only code you have to change here is to provide the location of the installers for IE and Netscape for Windows and Mac by replacing the text [URL_OF_DOWNLOAD] with the appropriate URL; note the download could come from an FTP or HTTP web server and would link directly to the .exe or .sit installer file.

```
<SCRIPT LANGUAGE="JavaScript">
  // Note that you have to provide the full [URL_OF_DOWNLOAD]
  // where indicated below; can be ftp or http location
  function getPage()
  {
    var BrowserName = getBrowser();
    var PlatformName = getPlatform();
    var NetscapeURL = "[URL_OF_DOWNLOAD]/omwebns.exe";
    var ExplorerURL = "[URL_OF_DOWNLOAD]/omwebie.exe";
    var MacURL = "[URL_OF_DOWNLOAD]/omweb.sit.hqx";
    if (BrowserName == "IE" && PlatformName == "W")
    {
      var URLAddress = ExplorerURL;
    }
    else if (BrowserName == "IE" && PlatformName == "M")
    {
      var URLAddress = MacURL;
    }
    else if (BrowserName == "NS" && PlatformName == "W")
    {
      var URLAddress = NetscapeURL;
    }
    else if (BrowserName == "NS" && PlatformName == "M")
    {
      var URLAddress = MacURL;
    }
    else if (BrowserName == "NS" && PlatformName == "L")
    {
      var URLAddress = NetscapeURL;
    }
    else
    {
      var URLAddress = "ERROR";
    }
    return URLAddress;
  }
  function getBrowser()
  {
    {
      if (navigator.appName.indexOf("Explorer") != -1)
      {
        var BrowserName = "IE";
      }
      if (navigator.appName.indexOf("Netscape") != -1)
      {
```

```

var BrowserName = "NS";
}
return BrowserName;
}
function getPlatform()
{
if (navigator.appVersion.indexOf("Macintosh") != -1)
{
var PlatformName = "M";
}
if ((navigator.appVersion.indexOf("Win") != -1) ||
(navigator.appVersion.indexOf("Win95") != -1) ||
(navigator.appVersion.indexOf("Win98") != -1) ||
(navigator.appVersion.indexOf("WinNT") != -1))
{
var PlatformName = "W";
}
if (navigator.appVersion.indexOf("Linux") != -1)
{
var PlatformName = "L";
}
return PlatformName;
}

function Start()
{
var BrowserPlatformURL = getPage();
if (BrowserPlatformURL == "ERROR")
{
// document.write("This is not a supported browser/platform");
}
else
{
location = BrowserPlatformURL;
}
}
</SCRIPT>

```

Note the above javascript downloads the appropriate web client installer, it does not auto-install the web client files, rather the user then has to run the installer. To install the web client automatically on the user's machine, you need to create some kind of auto-installing file, such as a CAB file under Windows.

Automatic Installation

It is possible to allow users to download and fully install the web client automatically, when they first visit a web page containing the web client plug-in. This automatic installation of the web client is enabled using special installers that you have to place on your Omnis Server. These automatic installers are currently available for Windows only.

Configuring your Omnis Web Server to support Automatic Installers

To take advantage of the automatic installers, you need to use an html template file called "download.htm", available in the Studio tree. This template provides enhanced functionality for the automatic download and installation of the web client. If you want to use this template for local testing, rename the current template file "template.htm" in your Omnis html directory to template.old and rename "download.htm" to "template.htm". Having done this, any html generated as the result of local testing will be enabled for Automatic Installation. If you have existing html that you wish to update, you can copy the Netscape JavaScript function downloadNow() from the "download.htm" template and place it at the appropriate place in your html page. If your page supports ActiveX controls, you will also need to modify the ActiveX plugin declarations to specify a codebase. An example of this can be found in the "download.htm" template.

By default, the "download.htm" template uses default URLs to point to automatic installers which reside on the Omnis Server. If you wish to store these on your own server, you should copy the appropriate files from the installation CD and update "download.htm" to point to your URLs. The installer files are as follows:

- ☐ omwebie.cab
the Internet Explorer ActiveX installer.
- ☐ omwebns.jar
the Netscape Plug-in installer.

Chapter 10— Troubleshooting Guide and FAQ

This section contains notes and FAQs that hopefully solve some of the problems you may be having regarding the Omnis web client.

General

Q: I have an existing Omnis Studio application. Do I need to start from scratch to take advantage of the Omnis web client, or can I adapt my existing application?

A: You can adapt your existing application to use the Omnis web client to give parts of your application a web interface, you do not need to start from scratch. Developing an Omnis web application using the Omnis web client is very similar to designing any other Omnis application. You have to create remote forms, instead of window classes, you use the web components rather than the built-in fields and other external components, and you have to pay special attention to where your methods are to execute, i.e. either on the client or the server; apart from these considerations, developing a web application using the Omnis web client is the same as designing any other type of Omnis application.

Furthermore, if you have an Omnis 7 application, you can convert it to Studio and adapt it for web access using the Omnis web client.

Remote Forms

Q: I've added an event handling method behind a button, but when I test the form and click on the button, nothing happens, the method does not appear to run. Why is this?

A: It's probably because you haven't enabled the `evClick` event for the pushbutton. By default, all events for all web components are disabled; you have to enable individual events for an object in its `$events` property.

Q: Should I worry about my Windows users using small or large fonts?

A: No, web components use TrueType fonts only, and the client browser scales the font size at runtime, depending on what the client machine has set.

Q: I want a particular field to get the focus when my remote form is opened. How can I do this?

A: Specify the number of the field, set in the \$order property of the field, in the \$startfield property of the remote form.

Q: My remote form contains a pushbutton that has an icon, but when I test the form the icon does not appear. Why is this?

A: Web components, such as pushbuttons, checkboxes, and radio buttons, that use icons from either the Omnispic or Userpic data file, or #ICONS, must include the relevant icon page in the \$iconpages property of the form. For example, if a button uses an icon from the 'Browsers' icon page in the Omnispic icon data file, you must include the Browsers page name in the \$iconpages property. Note that this applies to literally any component that contains an image, including icon arrays, tree lists, sidebars, and so on.

Browser

Q: I have designed my remote form, but when I press Ctrl/Cmnd-T to test the form, my web browser does not display the form. Why is this?

A: Have you installed the web client? You must install the Omnis web client to design and test your remote forms while using the development version of Omnis. If the web client is not installed and you hit Ctrl/Cmnd-T to test the form, your web browser will not display the form. To fix this, install the web client using one of the installers on the Studio CD.

If you have installed the web client and still don't get to see the form, this may be because the ActiveX is not registered properly. The installer registers the ActiveX automatically, but if this has failed, for some unknown reason, you can do it manually using the REGSVR32.EXE program. Under Windows, open the Run dialog from the Start button, and enter the following command, using the correct path to the .DLL:

```
regsvr32.exe c:\windows\Webclient\Orfc.dll
```

Q: Ctrl/Cmnd-T shows two forms with Internet Explorer. Why is this?

A: If you have Netscape Navigator on your machine as well as Internet Explorer (IE) and you have installed both the ActiveX and Netscape plug-in, IE may try to use the Netscape plug-in from Navigator and display both objects. Ctrl/Cmnd-T uses a template file in the HTML folder to create a test HTML page. This file has the html source for both the Netscape and Explorer objects to allow testing for both browsers. To stop this, try removing the source for either the ActiveX or Netscape object from the **template.htm** file, deleting any HTML files already created by Ctrl-T in the HTML folder (do not delete template.htm), and then retest using Ctrl/Cmnd-T.

Q: When I try to test my form Omnis says “you must set a remote task for the remote form?”

A: All remote forms require a remote task, even for testing purposes in design mode; the task handles the connection between the Omnis Server and remote form on the client. So to test a remote form, you must specify a remote task for the form; you can do this in the \$designtaskname property of the remote form. All the form wizards ask you to specify a remote task, but the New Remote Form template available in the Component Store does not create a remote task or set the \$designtaskname, so you have to do this manually.

Q: Why do I get the “unable to locate class” message?

A: Either the name of your library and the OmnisLibrary parameter in your HTML file do not match, or your library is not open; the Omnis runtime and your library must be running at all times to allow remote access from a browser.

Omnis Server

Q: Do I set the Omnis serverport property to 80?

A: No! Omnis should not be set to port 80, or any other number or service name already in use. Omnis needs the service of a standard web server. Therefore set the serverport in Omnis to a relatively high number, containing at least four digits, although you may need your MIS dept to allocate a number. And remember, the server port number in Omnis should be the same as specified in the OmnisServer parameter in your HTML file(s).

Q: Can I serialize the Omnis runtime running my web application, using my existing multi-user serial number?

A: No. You must serialize the Omnis runtime using a special web serial number. These numbers contain a “W” and allow up to a specified number of concurrent users. Contact your local sales representative to purchase a serial number.

Q: Why do I get a “server busy” message?

A: There are too many users connected to the server, the server is already full. You may also get this message on a development version if the browser connection is not closed correctly. In this case, Omnis thinks users are still connected. Such connections will eventually timeout, but to get around this, you can close and re-open your library.

Index

#ICONS, 69, 76

\$24hour, 83

\$action, 89, 116, 121

\$aligncolumns, 93

\$alignheadings, 93

\$allowclipboard, 106

\$arealist, 98

\$autotablen, 87

\$blocks, 118

\$boldover, 128

\$borderh, 87

\$borderv, 87

\$buttonbackground, 99

\$buttonstyle, 89, 119

\$cachepicture, 109

\$changeform(), 77

\$clearcachedforms(), 75

\$clientaddress, 138

\$closeform(), 77

\$columncount, 121

\$connectbytessent, 137

\$connectionid, 139

\$connectiontime, 139

\$construct(), 136

\$construct, Remote tasks, 142

\$currdays, 80

\$currentcursor, 98

\$currentid, 98

\$currentname, 98

\$cursor, 70, 119

\$dataname, 41, 85, 86, 89, 99

\$defaultnodeicon, 129

\$designtaskname, 169

\$digital, 83

\$disolvesize, 87

\$displayformat, 87, 93

\$effect, 128

\$enabled, 86

\$enterable, 129

\$event(), 40, 70

 Tasks, 137

\$events, 70, 91, 119, 130

\$events property, 167

\$expandcollapse, 129

\$expandcollapseicon, 129

\$fadeondatachange, 87

\$fadestyle, 87

\$fast, 106

\$fieldstyle, 86, 106

\$filecreator, 90

\$filename, 89

\$flashonclick, 98

\$frameonenter, 98

\$horizontal, 121

\$iconface, 83

\$iconid, 83, 89, 119

\$iconpages, 70, 89, 98, 99, 119, 128, 129, 168

\$imageheight, 106

\$imagewidth, 106

\$insideicon, 128

\$invertonenter, 98

\$issecure, 139, 160

\$lastresponse, 139

\$listcolumn, 85

\$listname, 85

\$max, 118, 123

\$maxtime, 138

\$maxusers, 137

\$maxvalue, 121

\$message, 108

\$min, 118, 123

\$minvalue, 121

\$moviefile, 120

\$movieurl, 120

\$multipleselect, 99

\$nodeiconspos, 129

\$noscale, 106, 109

\$nosmooth, 106

\$openform(), 77

\$order property, 168

\$outsideicon, 128

\$paction, 110

\$palette, 106

\$passwordchar, 86

\$path, 92

- \$pdatastream, 112
- \$picturealign, 92
- \$preadblocksize, 112
- \$preadstream, 112
- \$preadtimeinterval, 112
- \$progresscolor, 118
- \$redraw(), 74
- \$reportdata, 117
- \$reporttitle, 117
- \$reqcurbytesreceived, 138
- \$reqcurbytessent, 138
- \$reqmaxbytesreceived, 138
- \$reqmaxbytessent, 138
- \$reqtotbytesreceived, 138
- \$reqtotbytessent, 138
- \$requests, 138
- \$scaleicon, 83
- \$senddata(), 73, 106
- \$serverport, 158
- \$serverstacks, 145, 158
- \$showhscroll, 117
- \$showmessage(), 74
- \$shownodeicons, 129
- \$showstatusbar, 117
- \$showtext, 99
- \$showtoolbar, 117
- \$showurl(), 74
- \$showvscroll, 117
- \$smallicons, 99
- \$smalltextwidth, 99
- \$speed, 108
- \$startfield property, 168
- \$steps, 108
- \$strbutt..., 117
- \$stretch, 87
- \$strjobdlg..., 117
- \$strstatus, 117
- \$text, 119
- \$textcolor, 86, 129
- \$timeout, 138
- \$timeslice, 158
- \$timezone, 83
- \$timezoneadj, 83
- \$title, 90
- \$typelist, 90
- \$usepalette, 92
- \$val, 118, 123
- \$writejpeg(), 106
- \$zoomon, 117

- 80, port setting, 46, 157
- 80, server port, 169

- ActiveX, 13, 153
 - Html, 153
- Amend Startup Task option, 134
- Apache, 15
- Apache module, 15, 47
- Apache Server Extension, 159
- Autodetecting browser type, 156

- Begin critical block command, 147
- Border component, 80
- Breakpoints, 43
- Browser, 13
 - Detecting type, 156

- Caching, 75
- Calendar component, 80
- CGI, 15
 - nph-omniscgi.exe, 157
- cgi-bin folder, 159
 - web server, 15
- classid, 153
- Client, 12, 13
 - Web Client, 13
- Client connections, 138
- Client method execution, 76, 119
- Combo box component, 85
- Component Store, 78
 - Creating form controls, 78
 - Creating remote task classes, 132, 133
 - Creating task classes, 136
- Components
 - Form controls, 16
 - Web client, 161
- Components, Remote form, 78
- Componet Store, 41
- Connections, 14
- Connections pane, Monitor, 134
- Controls, Remote form, 78
- Cookies, 100
- Create New Monitor Task option, 132
- Ctrl/Cmnd-T, 15, 31
- ctrlmgr.dfl, 162
- ctrlmgr.lbs, 162
- currentpage property, 108
- Cursors, 70

- DAM interface*, 146
- Data handling
 - Optimizing, 73
- Date fields, 87, 93
- Debugging, 43
- Deploying remote forms, 44
- Deploying your Omnis web solution, 152
- Droplist component, 85

- Edit field component, 86
- End critical block command, 147
- evAfter, 71
- evAreaClicked, 98
- evBefore, 71
- evBusy, 137, 138
- evClick, 71, 85, 119, 167
- evDataRead, 111
- evDataReading, 111
- evDataWriting, 111
- evDataWritten, 111
- evDoubleClick, 71
- Event handling methods, 40
- Events, 14, 40, 70
 - Enabling, 167
 - Remote tasks, 137
- evHoursChange, 83
- evIconPicked, 40, 71
- evIdle, 137, 138
- evMinutesChange, 83
- evPortClosed, 111
- evPortError, 111
- evPortNames, 110
- evPortOpened, 111
- evPost, 137
- evRejected, 137
- evSecondChange, 83
- evSetPicked, 71
- evUserDataInit, 100
- Execute On Web Client option, 76
- External components
 - Form controls, 16

- Fade component, 87
- FAQs, Web Client, 167
- Form caching, 75
- Form fields, 33
- Form read/write component, 89
- FORMBACK, 80, 106, 161

- FORMCAL, 80
- FORMCLOCK, 83
- FORMFADE, 87
- FORMFILE, 89
- FORMFLDS, 82, 85, 86, 93, 107, 109, 119, 121, 161
- FORMGIF, 92
- FORMHPIC, 97
- FORMICON, 99
- FORMINFO, 100
- FORMJPEG, 105
- FORMMARQ, 108
- FORMPORT, 109
- FORMPRI, 116
- FORMPROG, 118
- FORMQT3, 120
- FORMROLL, 122
- FORMSBAR, 122
- FORMSLID, 123
- FORMTBAR, 125
- FORMTILE, 127
- FORMTIME, 127
- FORMTRAN, 128
- FORMTREE, 129
- FORMWASH, 130

- Gif component, 92

- Headed list component, 93
- History pane, Monitor, 134
- Hotpic component, 97
- HTML
 - ActiveX, 153
 - Netscape, 155
- Html files, 46
- HTML forms, 17, 140
- HTML Report task wizard, 134
- HTTP, 13
- https, 139, 160
- HTTPS, 13

- Icon array component, 99
- Icon Editor, 70
- Icon pages, 69, 76
- iconpages, 69
- Icons, 24, 69
- Info component, 100
- Inherit from Monitor task option, 132
- Installers, 161

- Creating your own, 162
 - Internet Explorer, 168
 - ISAPI, 15
 - ISP web hosting, 160
- Javascript, 156
- Jpeg component, 105
- kFFRead, 90
- kFFReadEntireFile, 90
- kFFReadEntireFileNow, 90
- kFFReadNow, 90
- kFFWrite, 90
- kFFWriteEntireFile, 90
- kFFWriteEntireFileNow, 90
- kFFWriteNow, 90
- Label component, 106
- Linux, 157
 - Web servers, 15
- List component, 107
- Load sharing, 149
- Load sharing mechanism, 150
- loopback address, 154
- Marquee component, 108
- Messages, 74
- Method execution, 14
- Method stacks, 144
- Methods, 34, 40
 - Client method execution, 76
- Monitor task wizard, 133
- Monitor window, 134
- Movie Player
 - Quicktime, 120
- Multiform Form Wizard, 66
- Multi-threaded server, 144
- Netscape, 153
- Netscape Navigator, 168
- Netscape plug-in, 13
 - Html, 155
- np_orfc.dat, 161
- np_orfc.dll, 161
- NT Servers, 158
- NTSERV.EXE, 158
- OK message command, 73
- Omnis Library, 16
- Omnis preferences
 - Server port, 158
- Omnis Runtime, 16
- Omnis server
 - Load sharing, 149
 - Multi-threaded, 144
 - Server usage, Monitor, 134
- Omnis Server, 16, 44
 - Commands, 146
 - Configuration, 157
 - Port number, 157
- Omnis web client, 13
- Omnis web server plug-in
 - Installation, 159
- Omnis.CFG
 - Port setting, 157
- Omnisapi.dll, 159
- OmnisClass parameter, 47, 154, 155
 - for direct access, 140
- OmnisLibrary parameter, 47, 154, 155
 - for direct access, 140
- Omnispic, 69
- OmnisPIC, 76
- OmnisServer parameter, 47, 154, 155, 169
 - for direct access, 140
 - Load sharing, 149
- Open window command, 74
- Open windows, 73
- Orfc.dat, 161
- Orfc.dll, 161
- Orfcgui.dll, 161
- Orfcmain.dll, 161
- Page pane component, 108
- pagecount property, 108
- Param1 to Param9 parameters, 154
- Password Form Wizard, 60
- Picture component, 41, 109
- Picture fields, 41
- Plain Form Wizard, 51
- Plain task wizard, 132
- Platform, autodetecting, 156
- pLineNumber, 85
- Plug-ins
 - Web server, 15
- Port component, 109
- Port number, 157
 - Windows NT, 157

- pParams, Remote tasks, 142
- Print component, 116
- Printing
 - on the client, 116
- Progress component, 118
- Prompt for data file, 44
- Prompt for data file command, 73
- Prompt for input command, 73
- Prompts, 73
- Properties
 - Web components, 79
- pUserData, 101
- Pushbutton component, 119
- Pushbuttons
 - Events, 167
- Quicktime component, 120
- Radio button component, 121
- Registering the Web Client, 15
- Remote form objects
 - Events, 70
- Remote forms*, 11, 49
 - Caching, 75
 - Controls, 78
 - Creating new, 69
 - Debugging, 69
 - Deploying, 44
 - Icons, 69
 - Multiple forms, 77
 - Programming, 72
 - Testing, 43
 - Tutorial, 19
 - Wizards, 49
- Remote task classes, 131
 - Creating new, 136
- Remote task wizards, 142
 - HTML Report, 134
 - Monitor, 133
 - Plain, 132
 - Submit, 135
- Remote tasks
 - \$construct, 142
 - Client access, 137
 - Events, 137
 - Instances, 136
- Roll button component, 122
- Runtime, 16
- scripts folder
 - web server, 15
- Scrolling text, 108
- Secure sockets, 139, 160
- Select an Icon dialog, 24
- Serial numbers, 169
- Server, 12, 15
- Server busy message, 169
- Server configuration, 158
- Server Configuration dialog, 158
- Server port, Setting the Omnis, 169
- Server Usage pane, Monitor, 134
- Servers
 - NT service, 158
- Sidebar component, 122
- Sidebar Form Wizard, 20, 62
- Sidebar with pages Form Wizard, 65
- Slider component, 123
- Start server command, 145, 146
- Stop server command, 145, 147
- Submit Form Wizard, 58
- Submit task wizard, 135
- Tab bar component, 125
- Tabs Form Wizard, 52
- Task properties, 137
- Tasks
 - Remote tasks, 131
- Templates
 - template.htm file, 168
- Threads, 146
- Tile component, 127
- Timeouts, 138
- Timer component, 127
- Transbutton component, 128
- Tree list component, 129
- Troubleshooting
 - Web client, 167
- Tutorial, 18
 - Remote forms, 19
- Use Existing Monitor Task option, 132
- Userpic, 69
- USERPIC, 76
- Variables, 34
- Wash component, 130
- Web Browser, 13

- Web client
 - Database access, 146
 - Installation, 168
 - Multi-threaded server, 144
- Web Client, 13
 - Installers, 161
 - Installing, 15
 - Registering, 15
- Web Client Components, 161
- Web components, 41, 78
 - Properties, 79
- Web server, 15
 - Configuration, 159
- Web Server, 46
- Web server plug-in, 15
 - Installation, 159
- WebServerScript parameter, 47, 154, 156
- WebServerUrl, 139
- WebServerUrl parameter, 154, 155
- WebServerURL parameter, 47
- Wizard Form Wizard, 57
- Working message command, 73
- Yes/No message command, 73
- Yield to other threads command, 148

How to use this manual



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

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

Bookmarks



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

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

Thumbnails



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

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

Browsing



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

Find

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

Search

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

To Search the Studio Index, select **Tools>>Search>>Indexes** to locate the Studio index (Index.pdx) on the Omnis CD. Next, select

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



Grabbing Text from the Screen



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

Getting Help

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