



Progress Dynamics. Web Development Guide

© 2005 Progress Software Corporation. All rights reserved.

Progress® software products are copyrighted and all rights are reserved by Progress Software Corporation. This manual is also copyrighted and all rights are reserved. This manual may not, in whole or in part, be copied, photocopied, translated, or reduced to any electronic medium or machine-readable form without prior consent, in writing, from Progress Software Corporation.

The information in this manual is subject to change without notice, and Progress Software Corporation assumes no responsibility for any errors that may appear in this document.

The references in this manual to specific platforms supported are subject to change.

A [Stylized], Allegrix, Allegrix & Design, Business Empowerment, eXcelon, ObjectStore, PeerDirect, Progress, Powered by Progress, Empowerment Center, Progress Empowerment Center, Progress Empowerment Program, Progress Fast Track, Progress OpenEdge, Progress Profiles, Partners in Progress, Partners en Progress, Progress en Partners, Progress in Progress, P.I.P., Progress Results, Progress Software Developers Network, ProVision, ProCare, ProtoSpeed, SmartBeans, SpeedScript, Technical Empowerment, and WebSpeed are registered trademarks of Progress Software Corporation or one of its subsidiaries or affiliates in the U.S. and/or other countries. AccelEvent, A Data Center of Your Very Own, AppsAlive, AppServer, ASPen, ASP-in-a-Box, BusinessEdge, Cache-Forward, Fathom, Future Proof, IntelliStream, ObjectCache, ObjectStore Event Engine, ObjectStore RFID Accelerator, ObjectStore Trading Accelerator, OpenEdge, POSSE, POSSENET, ProDataSet, Progress Business Empowerment, Progress for Partners, PSE Pro, PS Select, SectorAlliance, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, WebClient, and Who Makes Progress are trademarks or service marks of Progress Software Corporation or one of its subsidiaries or affiliates in the U.S. and other countries.

Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Any other trademarks and service marks contained herein are the property of their respective owners.

February 2005



Product Code: 4497
Item Number: 103614;V2.1B

Contents

Preface	ix
Purpose	ix
Audience	ix
Organization of this manual	x
How to use this manual	x
Typographical conventions	xi
Progress messages	xii
1. Introducing Web Application Development with Progress Dynamics	1-1
1.1 Overview of Web application development	1-2
1.2 Features that enable Web application development	1-2
1.2.1 Server-side features	1-3
1.2.2 Client-side features	1-4
1.2.3 Tool support	1-5
1.2.4 Web browser support	1-9
2. Progress Dynamics Web Architecture	2-1
2.1 Server-side architecture	2-2
2.1.1 Progress Dynamics and WebSpeed	2-2
2.1.2 How Progress Dynamics handles a Web request	2-4
2.2 Client-side architecture	2-11
2.2.1 Dynamic HTML	2-11
2.2.2 Client-side data management	2-12
3. Setting Up Progress Dynamics for Running Web Applications	3-1
3.1 The ICFWS session	3-2
3.1.1 Viewing the default ICFWS session settings	3-3
3.1.2 Modifying the default ICFWS session	3-8
3.1.3 Creating a new ICFWS session	3-15

3.1.4	Updating the XML configuration file	3-17
3.2	Configuring your Web server	3-19
3.2.1	Specifying the location of static files	3-20
3.2.2	Specifying the appropriate Progress Dynamics Broker.	3-21
3.2.3	Configuring the IIS Web server	3-22
3.3	Configuring the Apache Web server	3-26
3.4	Configuring the WebSpeed Broker and Agents	3-27
3.5	Testing the Web server	3-36
3.6	Testing the ICFWS session configuration	3-37
3.7	Starting an application development session	3-39
4.	Progress Dynamics Web Applications	4-1
4.1	Overview of Web application development	4-2
4.2	Running a Progress Dynamics Web application	4-3
4.3	URL syntax for Progress Dynamics Web applications	4-6
4.4	Considerations for development and use of Progress Dynamics Web applications	4-7
4.4.1	General considerations for Web look and feel	4-7
4.4.2	Development requirements and restrictions	4-8
4.4.3	Unsupported features and objects	4-12
4.4.4	Browser behavior	4-13
4.5	Deployment	4-15
5.	Customizing with Dynamic HTML	5-1
5.1	HTML	5-2
5.1.1	Default HTML file	5-2
5.1.2	Layouts	5-3
5.1.3	CSS links.	5-4
5.1.4	JavaScript links	5-5
5.1.5	Page title	5-5
5.1.6	XHTML conformance	5-5
5.1.7	Unicode UTF-8 support	5-6
5.1.8	Application help	5-7
5.2	Cascading Style Sheets	5-8
5.2.1	The default CSS files	5-9
5.2.2	Overview of CSS rules	5-9
5.2.3	Customizing with CSS.	5-12
5.2.4	Applying your own style sheets.	5-12
5.2.5	Applying themes for groups of users	5-14
5.3	JavaScript objects	5-14
5.3.1	Summary of JavaScript object files	5-15
5.3.2	Customizing with JavaScript	5-16
5.3.3	Extending existing JavaScript object types	5-17
5.3.4	Adding your own functions	5-19

5.4	Images	5-21
6.	Server-side Business Logic and Client Logic	6-1
6.1	Server-side business logic	6-2
6.1.1	Invoking server-side business logic from the client	6-2
6.1.2	Invoking JavaScript APIs from the server	6-3
6.1.3	Context management.	6-3
6.2	Client logic	6-5
6.2.1	Logic object	6-5
6.2.2	Object qualification	6-6
6.2.3	Client logic functions	6-7
A.	JavaScript API Reference	A-1
A.1	JavaScript actions	A-2
A.1.1	Syntax	A-2
A.1.2	Application (app)	A-3
A.1.3	Dialog (dlg)	A-3
A.1.4	Information (info)	A-4
A.1.5	Main (main)	A-6
A.1.6	Program (prg)	A-6
A.1.7	Server (server)	A-6
A.1.8	Tool (tool)	A-7
A.1.9	Utilities (util)	A-8
A.1.10	WebBusinessObject (wbo)	A-9
A.1.11	WebDataObject (wdo)	A-10
A.2	JavaScript for client-side functions and commands	A-12
A.2.1	SDO functions	A-13
A.2.2	TreeView commands and functions.	A-15
Index	Index-1	

Figures

Figure 1–1:	Dynamic Properties dialog box	1–6
Figure 1–2:	The Web Test Page	1–8
Figure 2–1:	Progress Dynamics and a WebSpeed configuration	2–3
Figure 2–2:	DHTML in Progress Dynamics	2–12
Figure 3–1:	ICFWS startup parameter	3–2
Figure 3–2:	AppServer and WebSpeed Broker Options	3–20
Figure 4–1:	An example of a Windows GUI application (oeCustBrowseWin)	4–3
Figure 4–2:	An example of a Web application (oeCustBrowseWin)	4–4
Figure 4–3:	Customer Orders Browse running in a Web browser	4–5
Figure 4–4:	Including displayed field in dynamic lookup retrieval	4–10
Figure 5–1:	The page layout in default.htm	5–3
Figure 5–2:	Default help topic	5–7
Figure 5–3:	CSS rule example	5–9
Figure 5–4:	StyleSheetFile in the Dynamic Properties dialog box	5–13
Figure 5–5:	The Dynamics Images utility	5–21

Tables

Table 1–1:	Supported browsers and platforms	1–9
Table 3–1:	Default values for ICFWS session properties	3–7
Table 3–2:	Directory structure for static files	3–21
Table 4–1:	Unsupported objects	4–13
Table 4–2:	TreeView characteristics in Web applications	4–14
Table 5–1:	JavaScript object files	5–15
Table 6–1:	List of client logic API functions	6–7

Preface

Purpose

This guide describes the features in Progress Dynamics® that allow developers to run Progress Dynamics applications in Web browsers. It also describes how to customize and adapt those applications to make them more suitable for display in a Web browser. In addition, this guide provides information on how to set up an application development environment to create and run Web applications.

Audience

This guide is designed for Web application programmers who are familiar with the Progress Dynamics application development environment. Knowledge of HTML, Cascading Style Sheets, and JavaScript is necessary to understand the material in this guide.

Organization of this manual

[Chapter 1, “Introducing Web Application Development with Progress Dynamics”](#)

Provides a general description of Web application development using Progress Dynamics.

[Chapter 2, “Progress Dynamics Web Architecture”](#)

Describes how the components of Progress Dynamics work together to run Web applications.

[Chapter 3, “Setting Up Progress Dynamics for Running Web Applications”](#)

Describes how to set up the Progress Dynamics environment, Progress®WebSpeed®, and Web servers in order to run Web applications.

[Chapter 4, “Progress Dynamics Web Applications”](#)

Describes how to create, run, and deploy Progress Dynamics Web applications.

[Chapter 5, “Customizing with Dynamic HTML”](#)

Describes how to use HTML, Cascading Style Sheets, and JavaScript, to customize Progress Dynamics for a Web browser environment.

[Chapter 6, “Server-side Business Logic and Client Logic”](#)

Describes how to use server-side business logic and client logic in Progress Dynamics Web applications.

[Appendix A, “JavaScript API Reference”](#)

Describes the JavaScript API, which can be used for customizing Web applications.

How to use this manual

Before you read this guide, you should complete the tutorial in [Getting Started with Progress Dynamics](#). When you complete the tutorial, you will have a number of example application objects that you can run as Web applications. You can also use the example application objects when experimenting with customizations. If you have developed your own Progress Dynamics application, you can experiment with running your application as a Web application, instead of using examples developed in the tutorial.

Typographical conventions

This manual uses the following typographical conventions:

- **Bold typeface** indicates:
 - Commands or characters that the user types
 - That a word carries particular weight or emphasis
 - Names of user interface elements
- *Italic typeface* indicates:
 - Progress variable information that the user supplies
 - New terms
 - Titles of complete publications
- Monospaced typeface indicates:
 - Code examples
 - System output
 - Operating system filenames and pathnames

The following typographical conventions are used to represent keystrokes:

- Small capitals are used for Progress key functions and generic keyboard keys.
END-ERROR, GET, GO
ALT, CTRL, SPACEBAR, TAB
- When you have to press a combination of keys, they are joined by a hyphen. You press and hold down the first key, then press the second key.
CTRL-X
- When you have to press and release one key, then press another key, the key names are separated with a space.
ESCAPE H
ESCAPE CURSOR-LEFT

Progress messages

Progress displays several types of messages to inform you of routine and unusual occurrences:

- Execution messages inform you of errors encountered while Progress is running a procedure (for example, if Progress cannot find a record with a specified index field value).
- Compile messages inform you of errors found while Progress is reading and analyzing a procedure prior to running it (for example, if a procedure references a table name that is not defined in the database).
- Startup messages inform you of unusual conditions detected while Progress is getting ready to execute (for example, if you entered an invalid startup parameter).

After displaying a message, Progress proceeds in one of several ways:

- Continues execution, subject to the error-processing actions that you specify, or that are assumed, as part of the procedure. This is the most common action taken following execution messages.
- Returns to the Progress Procedure Editor so that you can correct an error in a procedure. This is the usual action taken following compiler messages.
- Halts processing of a procedure and returns immediately to the Procedure Editor. This does not happen often.
- Terminates the current session.

Progress messages end with a message number in parentheses. In this example, the message number is 200:

**** Unknown table name *table*. (200)**

Use Progress online help to get more information about Progress messages. Many Progress tools include the following Help menu options to provide information about messages:

- Choose **Help→Recent Messages** to display detailed descriptions of the most recent Progress message and all other messages returned in the current session.
- Choose **Help→Messages**, then enter the message number to display a description of any Progress message.
- In the Procedure Editor, press the **HELP** key (**F2** or **CTRL-W**).

Introducing Web Application Development with Progress Dynamics

This chapter gives a general description of Web application development with Progress Dynamics®.

The major sections in this chapter are:

- [Overview of Web application development](#)
- [Features that enable Web application development](#)

1.1 Overview of Web application development

With Progress Dynamics, the same Repository definitions that generate a Windows user interface can generate a Web browser user interface. Dynamics managers automatically translate the abstracted definitions stored in the Repository into Dynamic HTML (DHTML). The DHTML employs Cascading Style Sheets (CSS) and JavaScript objects to create full-featured user interfaces. The Web browser user interface can contain the wide variety of menus, tool bars, and many of the other graphical elements that you see in Windows GUI applications.

NOTE: In this guide, the term *Web application* refers to an application whose user interface is displayed in a Web browser. The only Web browser that Progress Dynamics Version 2.x currently supports is Internet Explorer Version 5.5 and later (Version 6.0 is recommended). See the [“Web browser support”](#) section for more information.

1.2 Features that enable Web application development

Progress Dynamics enables Web application development through a combination of the following:

- [Server-side features](#)
- [Client-side features](#)
- [Tool support](#)
- [Web browser support](#)

1.2.1 Server-side features

The sections that follow briefly describe the server-side features in Progress Dynamics that enable the development of Web applications.

Request Manager

The Request Manager is the single point of entry for all Web requests. It, in turn, calls the other Dynamics managers that apply security, generate or retrieve session information, interact with databases, and create the DHTML.

The Request Manager is one of the components of the Dynamics Broker/Agent that is added to a Progress® WebSpeed® Version 3.1D Transaction Server during a Progress Dynamics installation.

User Interface Manager

After retrieving session data from the Session Manager and layout data from the Repository Manager, the User Interface Manager generates the DHTML that satisfies a client's request.

Like the Request Manager, the User Interface Manager (UIM) is one of the components of the Dynamics Broker/Agent that is added to a WebSpeed Version 3.1D Transaction Server during a Progress Dynamics installation.

NOTE: The output from the UIM complies with the W3C XHTML standard. It is well formed and can be read by XML parsers, processors, and browsers. Also, see the “XHTML conformance” section in [Chapter 5, “Customizing with Dynamic HTML.”](#)

Open Client DataObject

The Open Client DataObject APIs are used by the UIM to allow the Progress Dynamics WebSpeed Agent, which is running stateless, to interact with an SDO. The APIs accept and produce delimited data strings, so no external temp-tables are necessary to communicate with the SDO.

Dynamic Call Wrapper

The Dynamic Call Wrapper provides a generic mechanism for the Progress Dynamics WebSpeed Agent to run any non-SDO business logic, which could include a procedure or function residing in any Progress procedure file. For Progress Dynamics Web applications, it is usually used when the JavaScript API is invoked on the server.

1.2.2 Client-side features

The sections that follow briefly describe the client-side features in Progress Dynamics that enable the development of Web applications.

Static files

A number of static files implement DHTML in Progress Dynamics. The static files do not exist in the Progress Dynamics Repository, but are installed either in a Web Server's Document Root directory or in *install_dir\tty\icf\ry*. (Note that although these files are stored on the server, they are sent to and run on the client.)

The static files include:

- **HTML layout file** — The HTML layout file (*default.htm*) provides a starter page layout that includes a menu bar, a tool bar, a tree view, a message area, and hidden frames for client-server communications.
- **Cascading Style Sheets** — Two CSS files (*rymain.css* and *ryapp.css*) contain default style settings that you can inherit, modify, or override.
- **JavaScript object files** — A number of JavaScript files make up a library of event handlers, methods, and properties that you can inherit, modify, or override.
- **Utilities** — A number of HTML files provide utilities, such as pop-up calculators and calendars, that you can embed in your application.
- **Images** — Images include a number of *.gif* and *.jpg* files that you can use for buttons and tools.

JavaScript API

The JavaScript API contains a variety of methods that allow you to duplicate or extend features found in a Windows-based GUI.

WebDataObject (WDO)

On the Web browser client, data is managed within the WebDataObject (WDO), a nonvisual object. The WDO is the client-side equivalent of a server-side SDO.

The WDO manages all visual screen objects that use its data. It manages copies of old values, as well as changed values, in order to know what to submit back to the server-side SDO for record updates.

The WDO facilitates client-side object linking, similar to Progress SmartLinks™ technology.

WebBusinessObject (WBO)

WBO is the primary object for sending data back to the server. It performs the following tasks:

- Setting appropriate target information.
- Packaging WDO data into hidden form fields.
- Assembling lists of server actions.

NOTE: Do not confuse the WBO with the SmartBusinessObject (SBO). The function of the SBO is to integrate a number of SDOs, whereas the WBO is primarily an object for transferring data to the server.

1.2.3 Tool support

The tools listed in this section provide support for Web application development. For more information about using these tools, see the *Progress Dynamics Developer's Guide*.

Object Generator

The Object Generator tool generates Progress SmartDataObjects™, SmartDataBrowsers™, SmartDataViewers™, and SmartDataFields™.

Dynamic Property tool

The Container Builder and the Progress Dynamics AppBuilder use property sheets for managing attributes associated with static and dynamic objects. The Dynamic Property tool maintains these property sheets and provides the ability to define:

- Attribute values by Result Code at the object master and object instance levels.
- HtmlClass, StyleSheetFile, JavaScriptFile, and JavaScriptObject attribute values for a container.
- HtmlClass attribute values for any object type, object master, or object instance.
- HTML events for a dynamic viewer field.

The Dynamic Properties dialog box maintains container and object attribute instance values. As shown in [Figure 1–1](#), it consists of a nonmodal window with a list of all the properties (attributes and events) that you can modify. If more than one object is selected, the dialog box displays only the properties that are common to all the objects. A change to a common property affects all the selected objects.

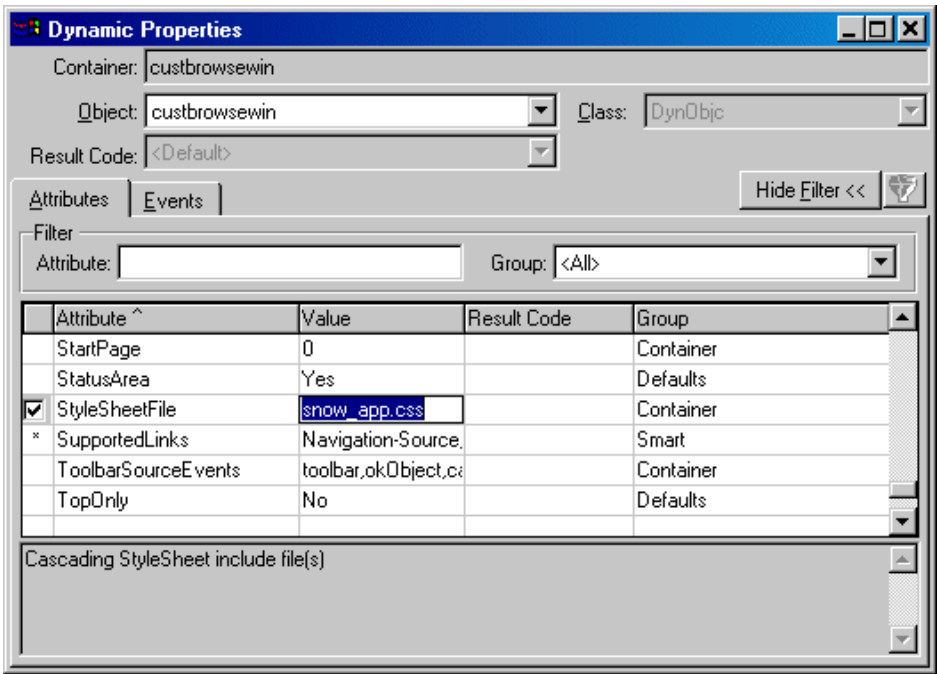


Figure 1–1: Dynamic Properties dialog box

The values for each property appear in an updateable browser control. The browser control lists the property, its value, result code, and group. You can either type directly into the value, select from a combo box, or have a lookup button that spawns a dialog box and returns a string value.

A toggle box located next to each property indicates which properties have been overridden. Deselecting the toggle box deletes the assigned value for the property. The property and group columns can be sorted by clicking on the column label. Every supported widget type is stored in the Repository as an object class. Various APIs exist to retrieve and save those attributes associated with a widget type, and to assign attribute values.

Container Builder

The Container Builder is used to create and manage Repository object data for complex dynamic containers. It maintains information about all objects on the container, the pages they are on, how they are linked, and their layout on their respective pages.

Toolbar and Menu Designer

The Toolbar and Menu Designer tool defines application toolbars and menus. The User Interface Manager reads the toolbar and menu definitions from the Dynamics Repository using the Repository Manager `getToolbarBandActions` API. For Web applications, the User Interface Manager adds two additional toolbar buttons for the **Filter** and **Find** dialog boxes.

AppBuilder

The Progress Dynamics AppBuilder creates and maintains many static and dynamic objects. Of particular interest for the Dynamics Web applications is the ability to define HTML events for viewer field objects using the new property sheet.

NOTE: When working with Dynamic Viewers in the AppBuilder, you should implement non-SDO fields as SDO calculated fields. This is due to the tight integration of SDOs into the Progress Dynamics framework and the DHTML client. Manipulating SDO calculated fields is far easier than manipulating non-SDO fields.

Web Test Page

The Web Test Page, shown in [Figure 1–2](#), contains a number of tests and utilities for checking your Progress Dynamics Web setup and configuration. It also contains links to lists of static objects, dynamic objects, and static images.

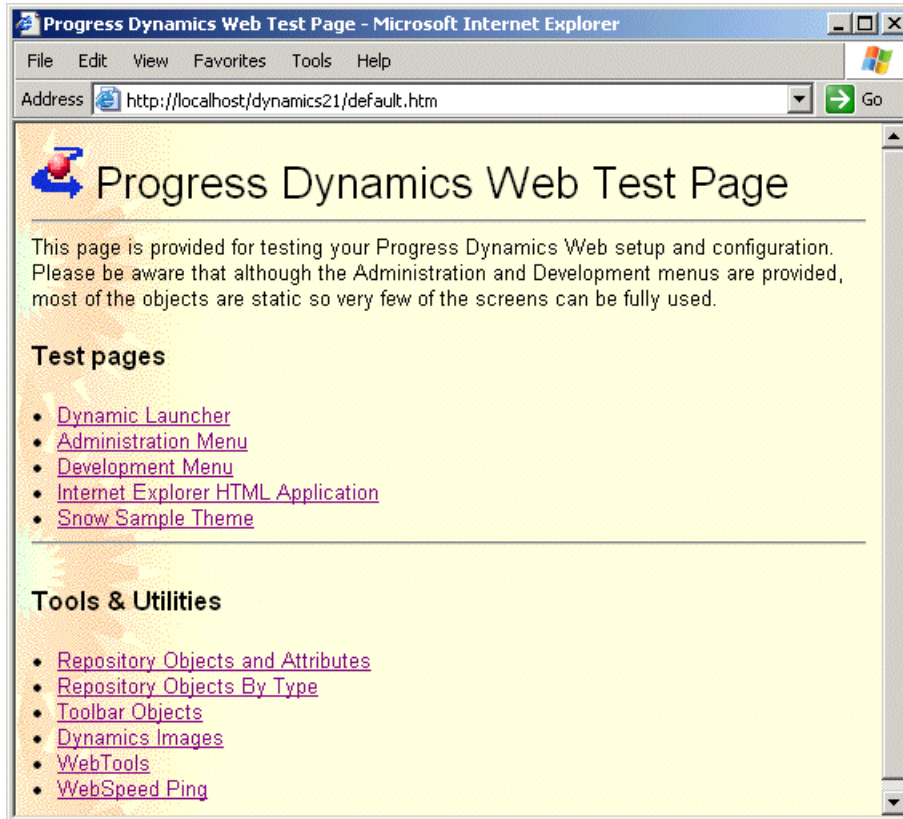


Figure 1–2: The Web Test Page

You can access the Web Test Page from a Web browser. The page is the `default.htm` file that is in `install_dir\tty\icf\ry\`. (The URL shown in [Figure 1–2](#), shows `dynamics21` as a virtual directory that points to `install_dir\tty\icf\ry\`.)

1.2.4 Web browser support

[Table 1–1](#) lists the browsers and platforms supported by Progress Dynamics.

Table 1–1: Supported browsers and platforms

Web browser	Windows	Linux	MAC
Mozilla 1.7	Yes	Yes	Yes
Microsoft Internet Explorer, Version 6.0	Yes	No	No

NOTE: Internet Explorer Version 6.0 running on Windows 2000, XP, NT, or 98 is recommended.

Progress Dynamics Web Architecture

This chapter provides an overview of how the components of Progress Dynamics work together to implement Web applications.

The major sections in this chapter are:

- [Server-side architecture](#)
- [Client-side architecture](#)

2.1 Server-side architecture

The server-side architecture of Progress Dynamics includes components for responding to requests from client browsers and for creating DHTML based on definitions stored in the Dynamics Repository.

The topics in this section that describe the server-side architecture are:

- [Progress Dynamics and WebSpeed](#)
- [How Progress Dynamics handles a Web request](#)

2.1.1 Progress Dynamics and WebSpeed

Progress Dynamics Web applications use the basic WebSpeed architecture. In [Figure 2–1](#), the WebSpeed components that are different in Progress Dynamics are shown in bold.

The Progress® WebSpeed® Messenger and NameServer are the same as those used in a standard WebSpeed configuration. In fact, as [Figure 2–1](#) implies, a standard WebSpeed configuration can coexist with a Progress Dynamics Web configuration. Both could share the same Messenger and NameServer.

However, the Progress Dynamics Broker and Agents differ from the standard WebSpeed Broker and Agents in the following ways:

- The Progress Dynamics Broker/Agent is configured with a different PROPATH setting, and with an ICFWS Session Type startup parameter. See [Chapter 3, “Setting Up Progress Dynamics for Running Web Applications,”](#) for more information about configuration.
- The Progress Dynamics Agent is actually composed of Dynamics managers. The Request Manager serves as the point of entry for Web requests. The User Interface Manager creates the DHTML that is sent to a client browser. The Progress Dynamics Agent depends on other Dynamics managers for various services (session management and security, for example).

For an overview of how the Progress Dynamics Agent operates, see the [“How Progress Dynamics handles a Web request”](#) section.

- The Progress Dynamics Agent does not run Web objects as a standard WebSpeed Agent does. In order to build a Web application, the Progress Dynamics Agent interacts with the Repository database to build application objects.

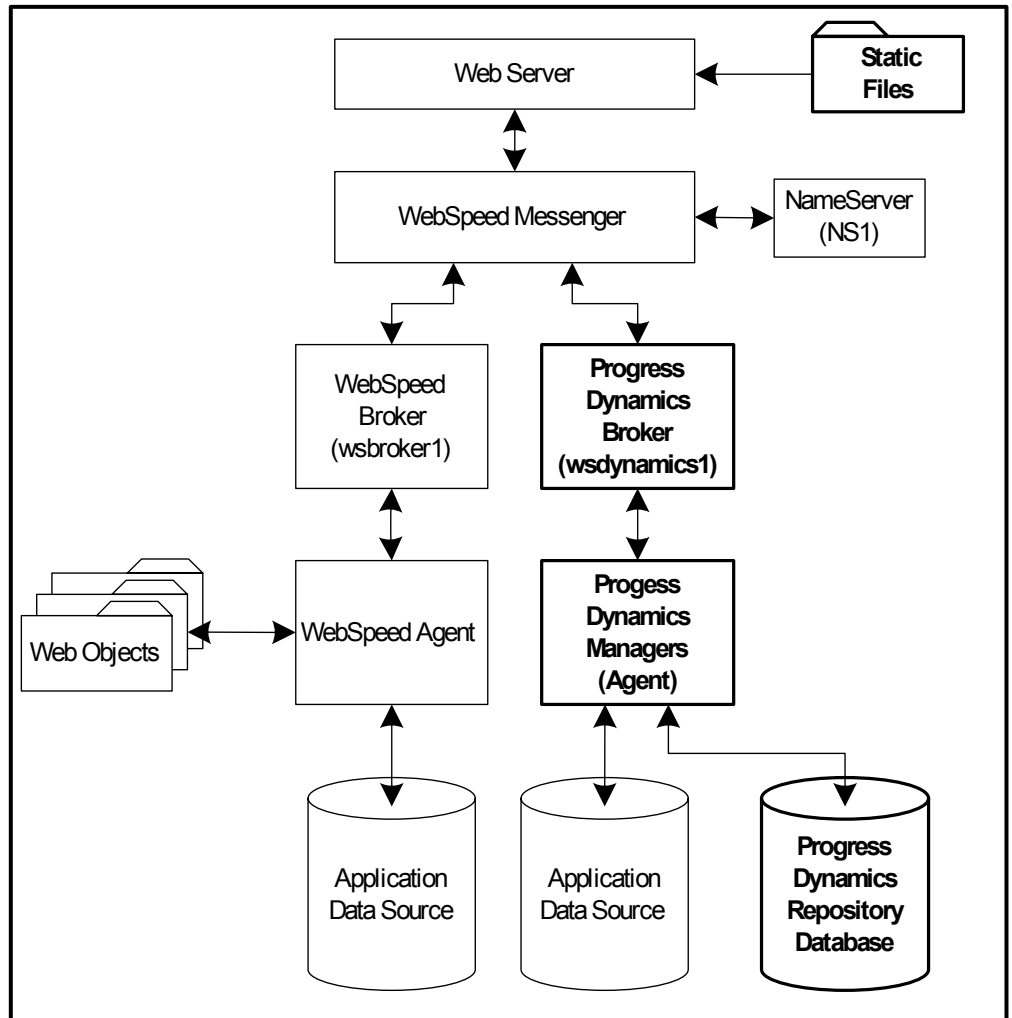


Figure 2–1: Progress Dynamics and a WebSpeed configuration

NOTE: Although the Progress Dynamics Broker/Agent differs internally and functionally from the standard WebSpeed Broker/Agent, you can still manage and configure the Progress Dynamics Broker/Agent with Progress Explorer. The Progress Dynamics Brokers are listed under the WebSpeed node in the Progress Explorer tree view.

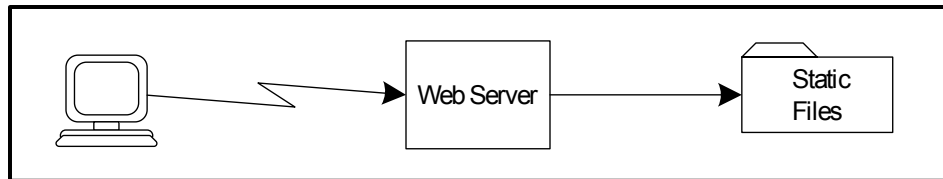
2.1.2 How Progress Dynamics handles a Web request

This section introduces you to the major Progress Dynamics components that are involved in processing a Web request and returning a DHTML page to a client browser.

The overall architecture is designed to maximize performance by reducing server hits. The DHTML page and data are sent separately, so only affected objects are updated, and slow updates of entire pages are avoided. Data caching is implemented on the client browser through JavaScript, so that updates can often be performed on the client side. In addition, updates for multiple SDOs are bundled on the client browser and sent to the server as a single request.

The following is a simplified illustration of a round-trip request between the client browser and the Progress Dynamics Agent:

1. The client browser initiates a request by means of a URL:



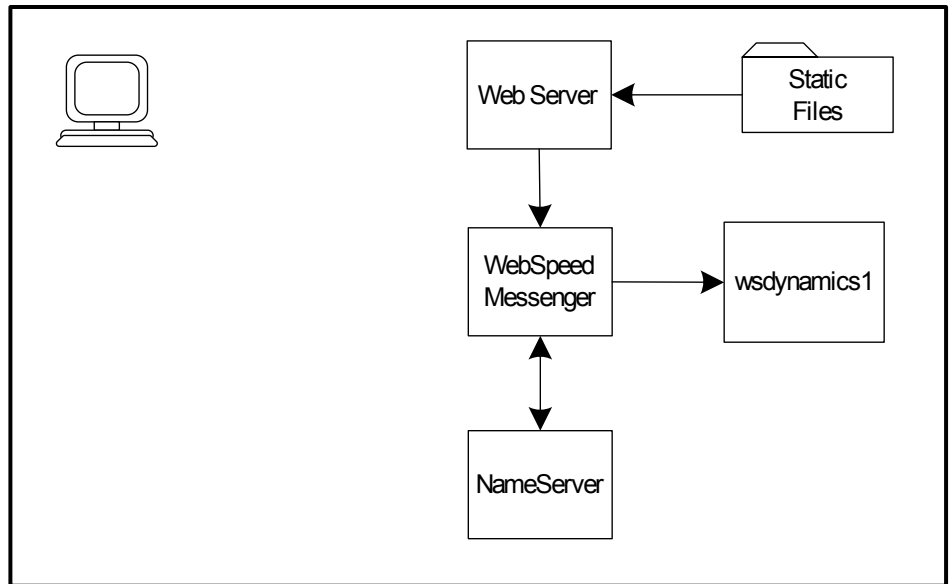
Typically, the URL contains:

- The Web Server's hostname.
- The path of the folder that contains the Progress Dynamics static files. The content of this folder includes HTML pages, Cascading Style Sheets, JavaScript files, and image files.
- The name of the startup Web page (usually `default.htm`).
- A name/value pair where the `icfobj` variable is set to the name of the Progress Dynamics application object.

The following shows the format of the URL:

```
//hostname/dynamics_static_files/dhtml/default.htm?icfobj=MyAppObj
```

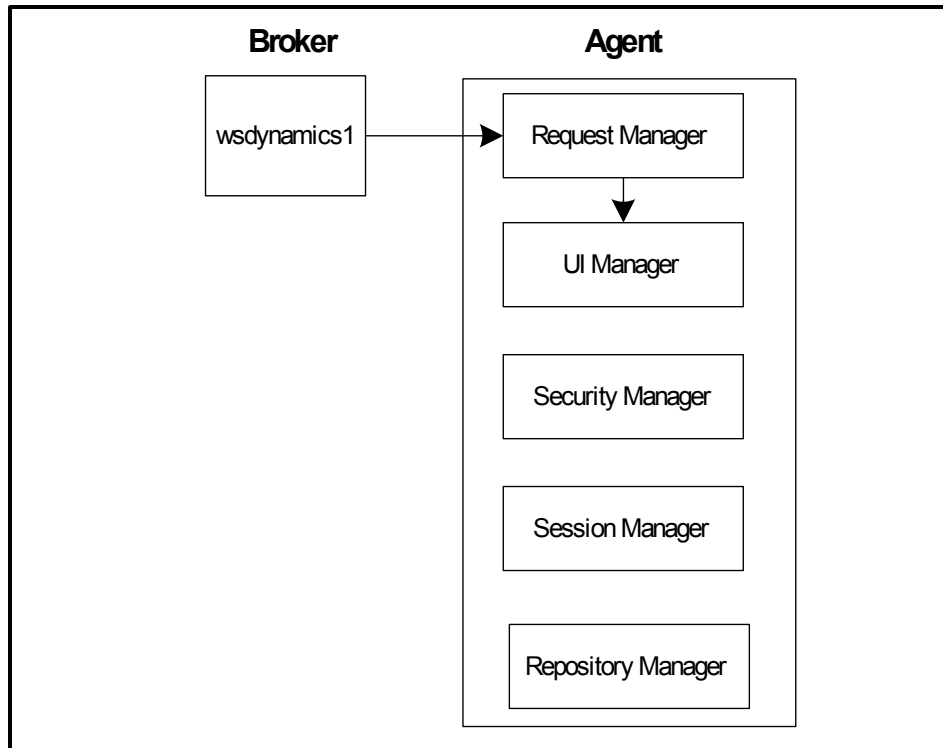
2. The `default.htm` file causes `ryinithtml.js` to run, which initiates a WebSpeed request as illustrated in the following diagram:



The WebSpeed request includes the name of a Progress Dynamics Broker (in this case the default, `wsdynamics1`) and the name of the Progress Dynamics application object.

The standard WebSpeed Messenger/NameServer architecture is used to find the Progress Dynamics Broker.

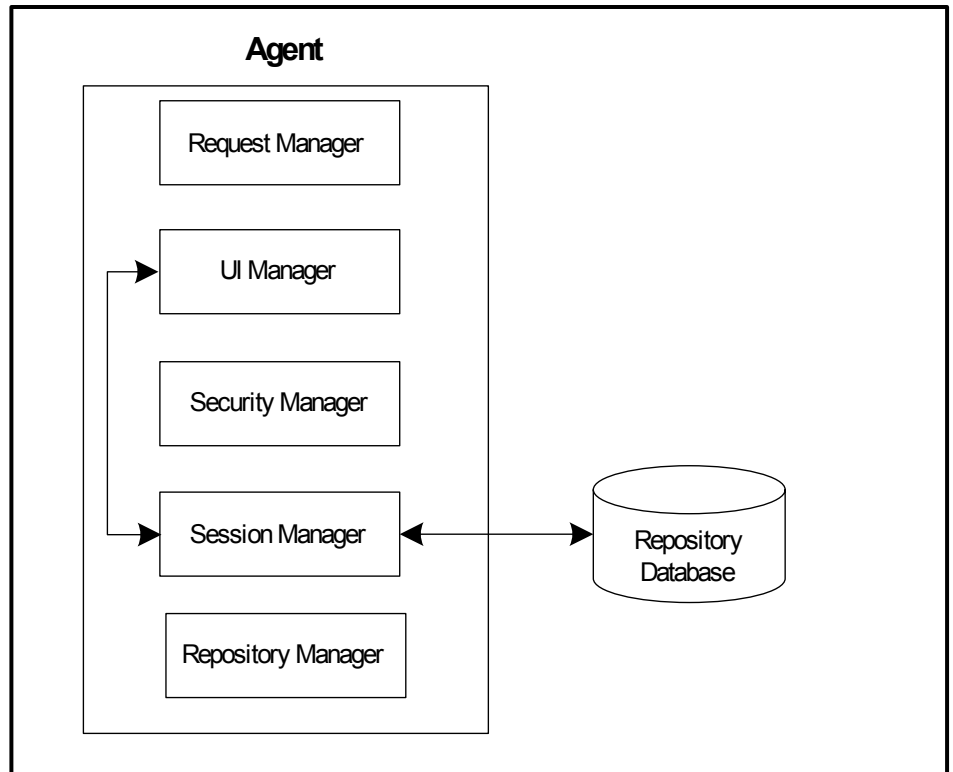
3. As in standard WebSpeed architecture, the Broker finds a free Agent and passes the request to it as shown:



Unlike the standard WebSpeed Agent, the Progress Dynamics Agent is comprised of Dynamics managers.

The Request Manager is the single point of for all Web requests received by the Agent. On initially receiving a Web request, the Request Manager determines the client UI type and then accesses the appropriate User Interface (UI) Manager.

4. The UI Manager extracts incoming data from the Web stream and sends it to the Session Manager to store in a context table in the Repository database as shown:

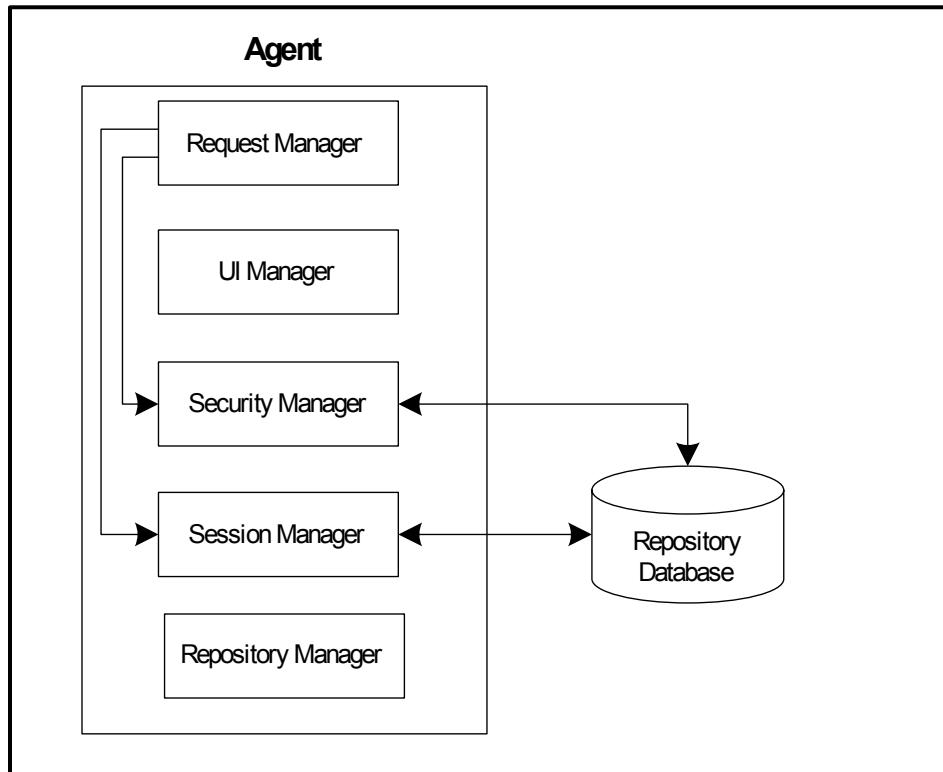


Although the Progress Dynamics Agent is running in the stateless mode, context can be maintained between requests because the Session Manager creates a unique ID for each session. This Session ID is passed back to the client. When the client sends another Web request, it includes a Session ID with the request. The Session Manager can match the Session ID with context tables and restore the context of the request. If there is no Session ID included with a Web request, the Session Manager creates a new one.

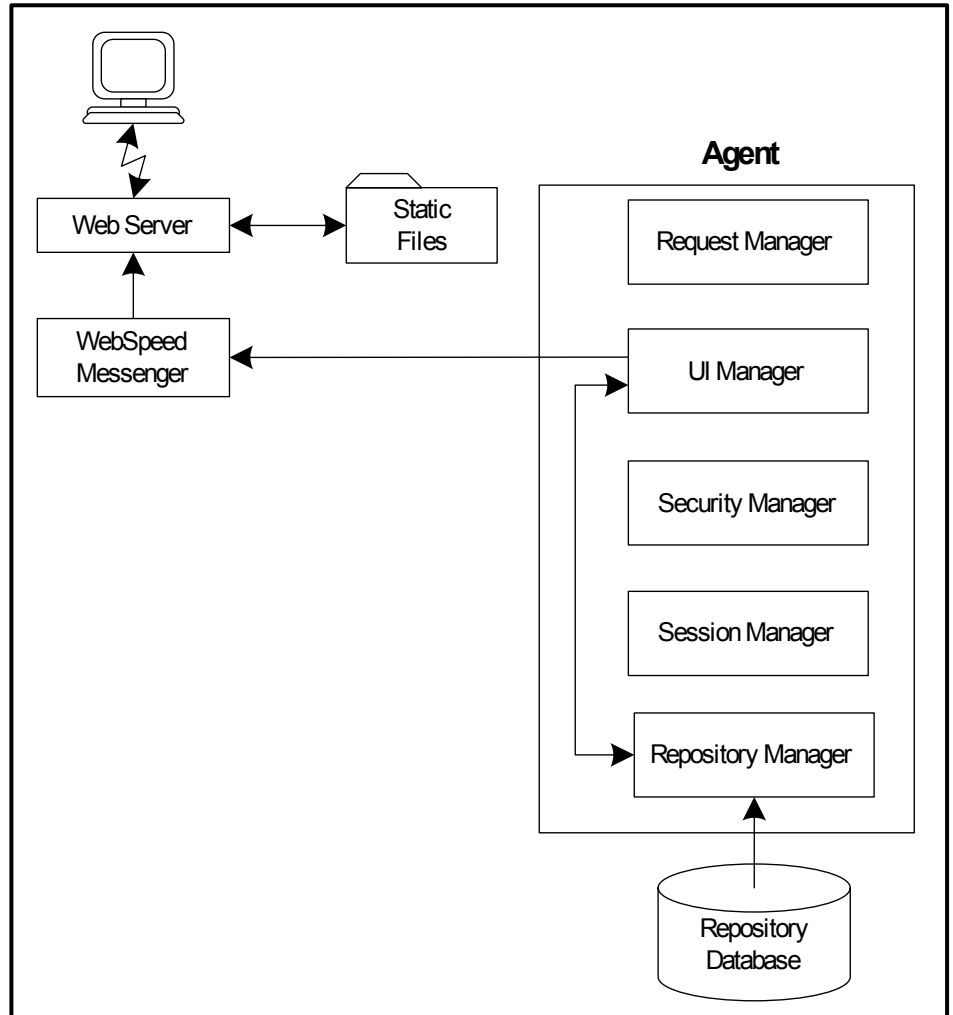
5. If a login is required at this point, the UI Manager sends a login page to the client. For Web browser applications, ICFWS session properties define login requirements. Use the Session Type Maintenance tool to modify ICFWS session properties. See the [“The ICFWS session”](#) section in [Chapter 3, “Setting Up Progress Dynamics for Running Web Applications.”](#)

Note again that the Agent is not locked while waiting for the client to log in. When the client does log in, the login information (user name and password) also includes a Session ID so context can be restored.

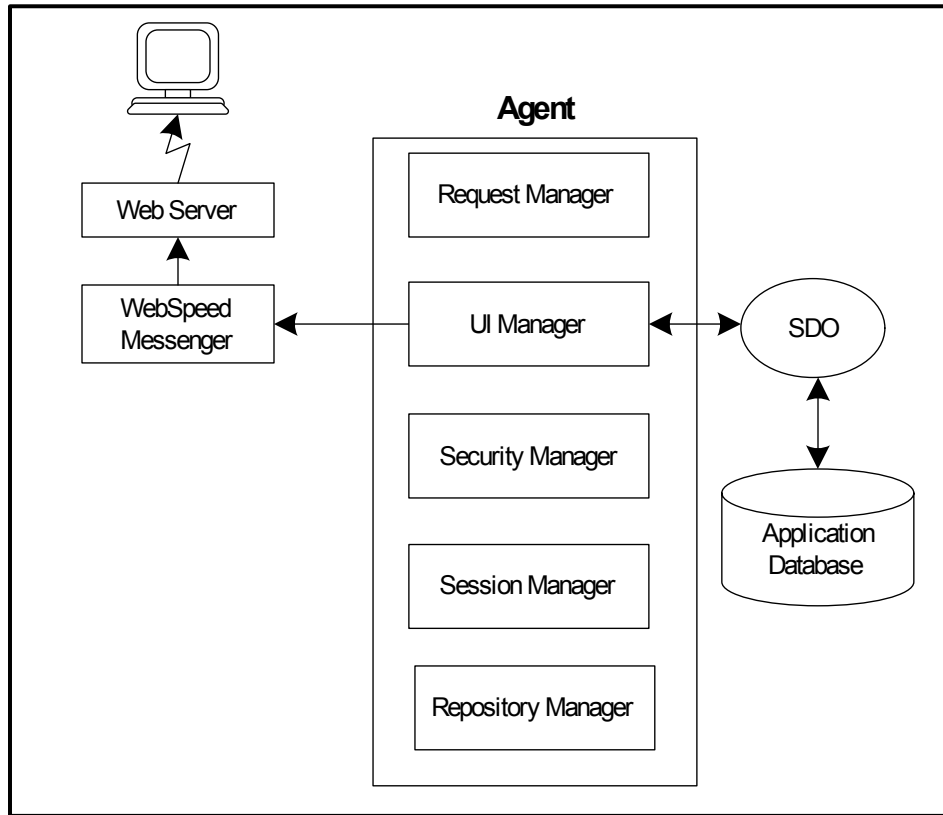
6. When login information is received, the Request Manager calls the Security Manager to authenticate the user. The Request Manager also calls the Session Manager to restore context as shown:



7. The UI Manager retrieves the application object information from the Repository Manager. The Web server accesses the static DHTML files (HTML layout, JavaScript, and Cascading Style Sheets) and images. Then the interface information is sent to the client as shown:



8. The UI Manager retrieves application data and sends it to the client as shown:



Notice that the UI Manager sends interface information and application data as two separate steps in response to a single request. This provides a performance advantage for subsequent updates. Only data for affected objects will be sent. The need to update the entire page is eliminated.

9. In the final stage, the UI Manager returns control to the Request Manager. The Request Manager does some flushing of context and other cleanup tasks. It is then available to handle another Web request.

2.2 Client-side architecture

The client-side architecture of Progress Dynamics Web applications is designed to provide a full-featured user interface. The architecture also includes features that allow for client-side processing and data storage, which improves performance by minimizing network traffic.

The topics in this section that describe the client-side architecture are:

- [Dynamic HTML](#)
- [Client-side data management](#)

2.2.1 Dynamic HTML

Dynamic HTML (DHTML) is a broad term that refers to a number of technologies that enable interactive Web pages. Progress Dynamics uses DHTML to create the objects (buttons, menus, toolbars, and so on) that comprise a full-featured user interface for applications running in a Web browser.

As shown in [Figure 2–2](#), DHTML in Progress Dynamics employs an HTML page that links to Cascading Style Sheets (CSS files) for formatting, and JavaScript files for implementing objects. These are static files located in *install-dir\tty\icf\ry\dhtml*. They are available to the UI Manager when it creates dynamic Web pages in response to client requests. See [Chapter 5, “Customizing with Dynamic HTML,”](#) for more information about the HTML pages, CSS files, and JavaScript files.

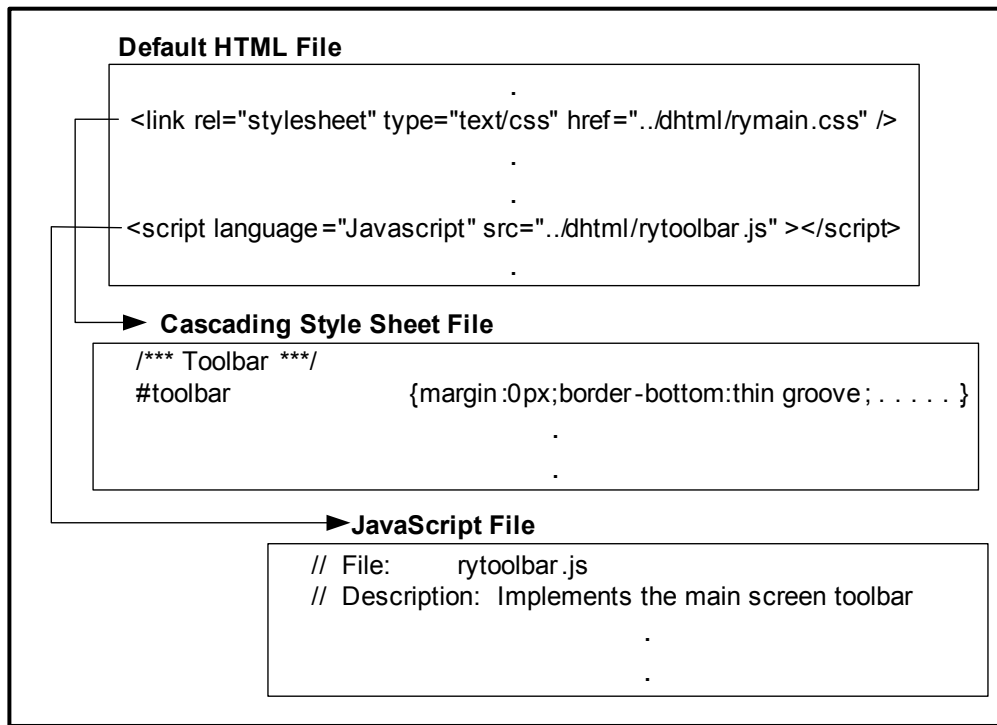


Figure 2–2: DHTML in Progress Dynamics

2.2.2 Client-side data management

User interface data, application data, and messages are stored and managed on the client to minimize network traffic to the server.

UI data

User interface data is stored with UI objects. For example, if a menu object contains multiple levels, the data for all levels is stored on the client. Therefore, when a user expands a menu selection, the response is immediate. The user does not have to wait for the event to be sent to a server for processing and updating.

Application data

WebDataObjects (WDOs) are used on the client side for managing application data. WDOs are proxies for server-side SDOs and are written in JavaScript. They provide a performance advantage when data manipulation can be performed on the client. For example, after doing a query and receiving all the records in the Customer table, you might want to perform a sort

based on balance due. Instead of requesting another query and sort from the server, you can do a local sort on the data that you already have on the client.

WDOs use a WebBusinessObject (WBO) for communication with the server. There is only one WBO per page, but there can be many WDOs associated with it. Both the WBO and WDOs exist within the Hidden Frame of the default HTML layout page (`default.htm`). For more information about `default.htm`, see the “Default HTML file” section in [Chapter 5](#), “Customizing with Dynamic HTML.”

Messages

An Info object is available to store and display messages. Messages are usually loaded immediately after login and can be displayed in a window area or in a pop-up. Therefore, a user error or some other condition can generate a local response rather than a network call to a Web server. Messages are displayed in the Message Frame of the default HTML layout page (`default.htm`). For more information about `default.htm`, see the “Default HTML file” section in [Chapter 5](#), “Customizing with Dynamic HTML.”

Also see the “Information (info)” section in [Appendix A](#), “JavaScript API Reference.”

Setting Up Progress Dynamics for Running Web Applications

This chapter describes how to set up the Progress Dynamics environment, WebSpeed, and Web servers in order to run Web applications. To illustrate procedures, it uses the application developed in the tutorial contained in *Getting Started with Progress Dynamics* as an example.

For more information, see the *Progress Dynamics Administration Guide*. Also see the *Progress Dynamics Installation Guide* for installation and basic setup information.

The major topics in this chapter are:

- [The ICFWS session](#)
- [Configuring your Web server](#)
- [Configuring the WebSpeed Broker and Agents](#)
- [Testing the Web server](#)
- [Testing the ICFWS session configuration](#)
- [Starting an application development session](#)

3.1 The ICFWS session

ICFWS is the session type used by the Progress Dynamics WebSpeed Agent. It defines:

- Required managers and the order in which they start.
- Session properties, including login parameters.
- Session services, which include the your application's database as well as your application's Repository (icfdb).

Both the ICFWS session type and a default Broker (named `wsdynamics1`) are created during the Progress Dynamics installation. [Figure 3–1](#) shows that the ICFWS session type is specified in Progress Explorer as a startup parameter for the WebSpeed Agent running under `wsdynamics1`.

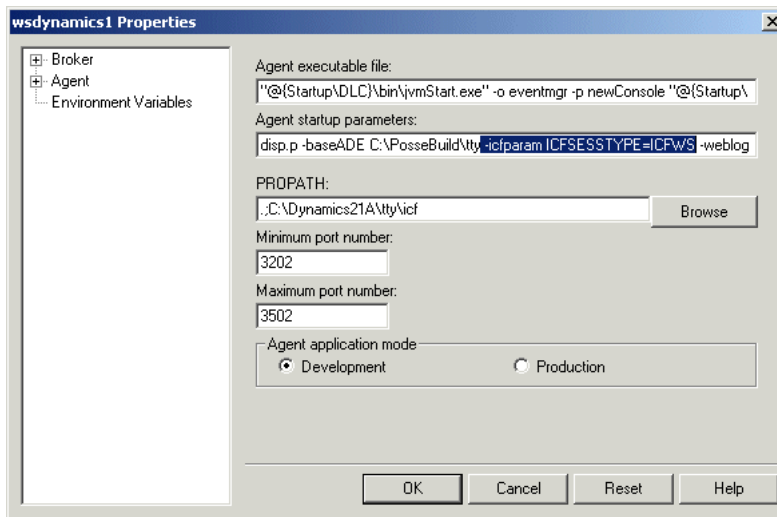


Figure 3–1: ICFWS startup parameter

For more information about session types, see the [Progress Dynamics Administration Guide](#).

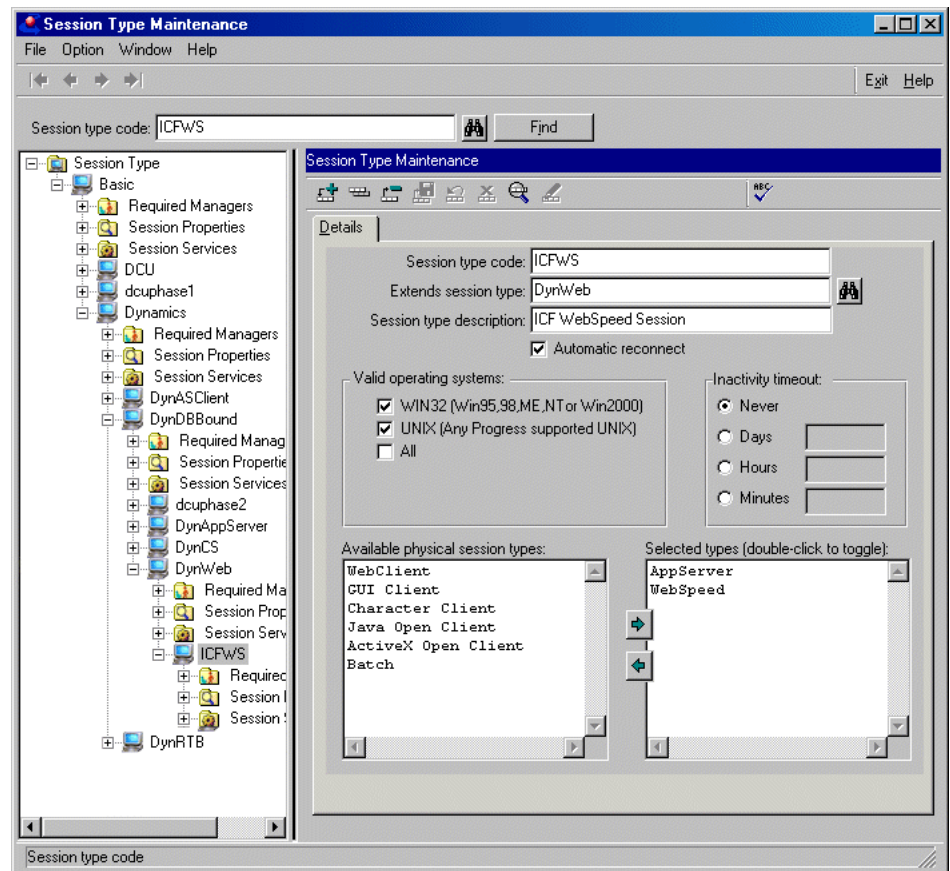
3.1.1 Viewing the default ICFWS session settings

To view the ICFWS session settings:

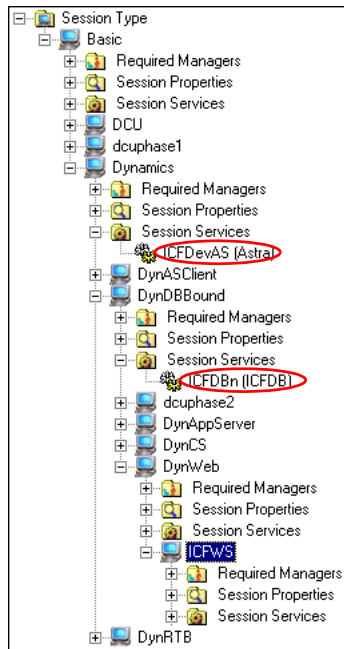
- 1 ♦ Start Progress Dynamics using a session that has administrative privileges.

If you intend to view the settings associated with the tutorial application, use the shortcut that you used to start a development session for the tutorial.

- 2 ♦ From the Progress Dynamics AppBuilder **Tools** menu, select **Administration**.
- 3 ♦ From the Administration **Session** menu, select **Session Type Control**.
- 4 ♦ In the **Session type code** lookup field at the top left of the window, enter **ICFWS** and click **Find**. The session type tree should expand to appear similar to the following:



- 5 ♦ The ICFWS session type inherits two required session services, **ICFDevAs** and **ICFDBn**. To verify that these services are present, expand the **Session Services** nodes under **Dynamics** and **DynDBBound**, as shown:



Note the following points:

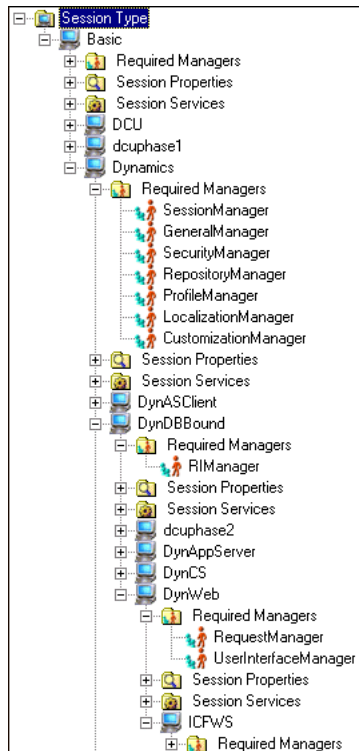
- **ICFDevAs** is an AppServer service. Although you do not use the AppServer to run Web applications, **ICFDevAs** is necessary to set an AppServer partition for SDOs.
- **ICFDBn** represents a network connection to the default Repository database. If you have created a Repository for your application, replace this entry with a service that represents a connection to the Repository for you application. See the [“Adding a database to ICFWS as a session service”](#) and [“Removing a session service from ICFWS”](#) sections.
- You need to add a session service for your application database. See the [“Adding a database to ICFWS as a session service”](#) section.

For more information, see the [“Modifying the default ICFWS session”](#) section.

- 6 ♦ The ICFWS session type inherits the following required managers, which you can verify by expanding the **Required Managers** node under the indicated parent nodes:

Parent node	Required managers
Basic	ConnectionManager
Dynamics	SessionManager
	GeneralManager
	SecurityManager
	RepositoryManager
	ProfileManager
	LocalizationManager
	CustomizationManager
DynDBBound	RIManager
DynWeb	RequestManager
	ManagerInterfaceManager

The expanded tree should appear similar to the following:



This listing represents the order in which the managers are started.

- 7 ♦ To view the properties defined for the ICFWS session type, expand the **Session Properties** nodes under the following session type nodes: **Basic**, **Dynamics**, **DynDBBound**, **DynWeb**, and **ICFWS**. The ICFWS type inherits all of the properties, overriding the values of two of them, `print_preview_preference` and `print_preview_stylesheet`.

- 8 ♦ Select a property to display its value. [Table 3–1](#) shows the default values for each of the properties.

Table 3–1: Default values for ICFWS session properties

Property	Default value
auto_dump_entity_cache	YES
DynamicsVersion	2.1B
UseThinRendering	No
session_type_template	yes
run_local	YES
allow_anonymous_login	NO
anonymous_user_name	anonymous
anonymous_user_password	(blank)
display_login_screen	Yes
session_time_format	HH:MM:SS
menu_images	disabled
image_path	ry/img,../img/
print_preview_preference	HTML
print_preview_stylesheet	../dhtml/webreport.css

See the [Progress Dynamics Administration Guide](#) for more information about session properties.

3.1.2 Modifying the default ICFWS session

To run your Progress Dynamics application as a Web application, the Progress Dynamics WebSpeed Broker/Agent must be able to access your application database and your application Repository. One method for enabling Broker/Agent access is to modify the default ICFWS session to include your application's database and Repository as session services. For an alternative method, see [“Creating a new ICFWS session.”](#)

This section describes how to modify the default ICFWS session, using the tutorial databases (described in [Getting Started with Progress Dynamics](#)) as examples.

The tutorial in [Getting Started with Progress Dynamics](#) uses DynSports, the Progress Dynamics sample database, as an application database. It is created in a work directory for the tutorial (<wrk>\Tutorial\databases\dynsports\dynsports.db).

A Repository for the application is created in <wrk>\Tutorial\databases\icfdb\icfdb.db.

In the tutorial, both databases are started in a script and run under servers in a network environment. For example:

```
call preserve "<wrk>\Tutorial\databases\icfdb\icfdb" -S icfdb
call preserve "<wrk>\Tutorial\databases\dynsports\dynsports" -S dynsports
```

Also note that icfdb and dynsports have been defined as services with port numbers in the Windows services file.

Adding a database to ICFWS as a session service

The general procedure for adding a database as a service in a session type is to define it as a logical service and as a physical service, and then add it to the session. You use the session tools found in the Administration tool to perform these tasks.

This section uses the DynSports database (created in the tutorial in [Getting Started with Progress Dynamics](#)) as an example of an application database. You can also use the procedure described in this section to add a Repository database to the ICFWS session.

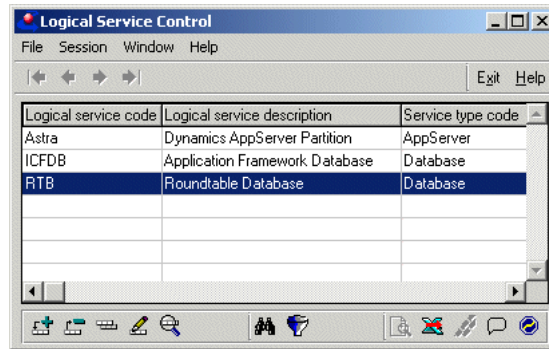
To add a database as a session service, follow these steps:


- 1 ♦ Start a Progress Dynamics session that has administrative privileges.

For the tutorial application, use the shortcut that you used to start a development session for the tutorial.

- 2 ♦ From the Progress Dynamics AppBuilder **Tools** menu, select **Administration**.

- 3 ♦ Choose **Session**→**Logical Service Control** from the Administration tool. The Logical Service Control dialog box appears:



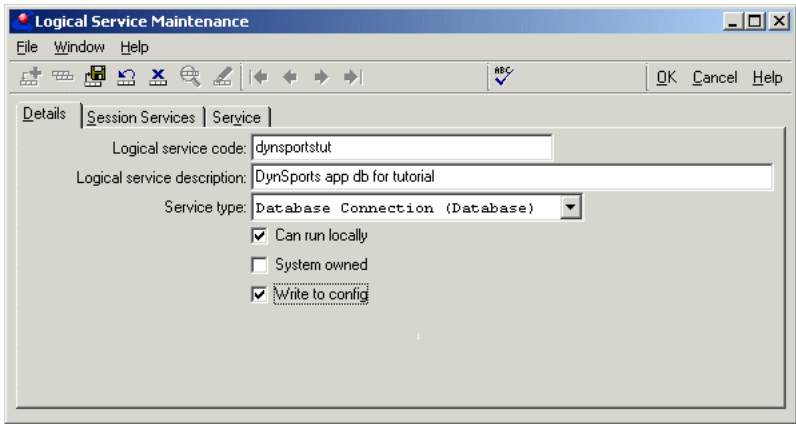
- 4 ♦ Choose **Add Record** from the **File** menu, or click  on the toolbar at the bottom of the window, to add a new service. The Logical Service Maintenance dialog box appears.
- 5 ♦ Set the values for the new service.

The following are typical values for DynSports:

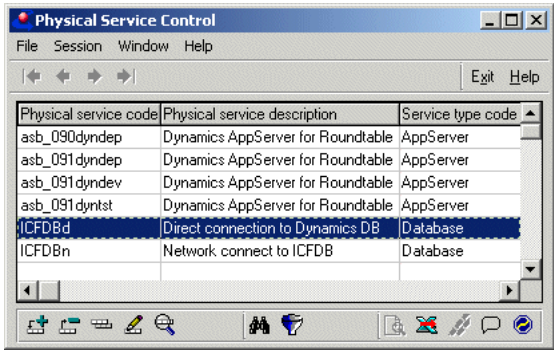
Field	Value
Logical service code	dynsportstut
Logical service description	DynSports app db for tutorial
Service type	Database Connection (Database)
Can run locally	Selected
System owned	Not selected
Write to config	Selected


NOTE: If you already created a dynports session service in the tutorial, you can use it instead of creating a new service called dynsportstut. However, if you have multiple instances of a sample database like DynSports or the ICFDB Repository database, it is useful to adopt a naming convention that identifies which database you are using. In this case, the name dynsportstut implies that the service uses the DynSports database that was created in the tutorial work directory.

- 6 ♦ Save the completed Logical Service Maintenance dialog box:



- 7 ♦ Exit from the maintenance and control dialog boxes.
- 8 ♦ Choose **Session→Physical Service Control** from the Administration tool. The **Physical Service Control** dialog box appears as shown:



- 9 ♦ Choose **Add Record** from the **File** menu, or click  on the toolbar at the bottom of the window, to add a new physical service. The **Physical Service Maintenance** dialog box appears.

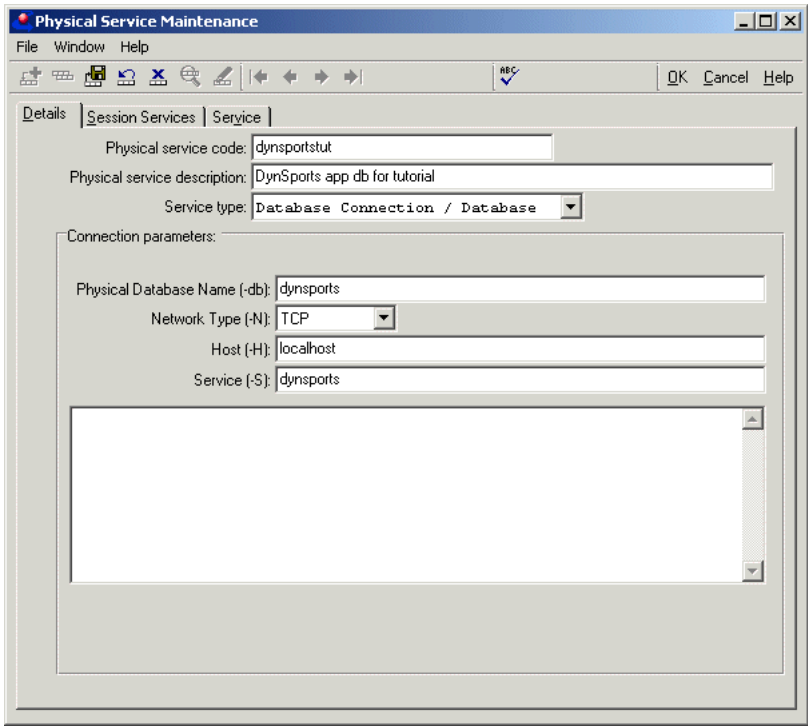
- 10 ♦ Enter the values for the new service. Assuming you start DynSports in multi-user mode, the following are typical values:

Field	Value
Physical service code	dynsportstut
Physical service description	DynSports app db for tutorial
Service type	Database Connection / Database
Physical Database Name (-db)	dynsports
Network Type (-N)	TCP
Host (-H)	localhost
Service (-S)	dynsports

The Service (-S) value, `dynsports`, implies that `dynsports` has been defined with a port number in the Windows services file and that you start the database with a command similar to:

```
proserve <wrk>\Tutorial\databases\dynsports\dynsports -S dynsports
```

11 ♦ Save the completed **Details** tab:



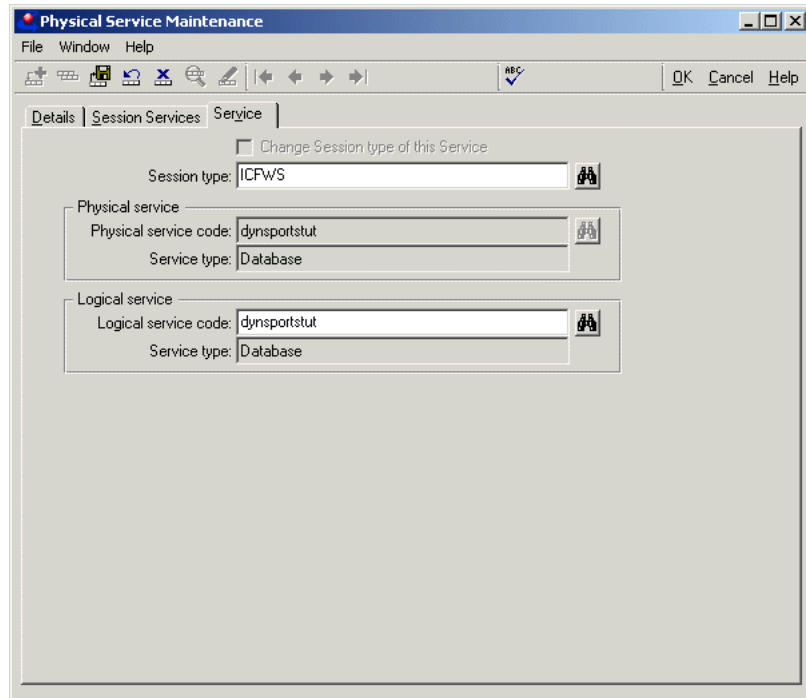
12 ♦ Select the **Service** tab in the **Physical Service Maintenance** dialog box and choose **Add record** ( on the toolbar).

13 ♦ Enter the following values:

Field	Value
Session type	ICFWS
Logical service code	dynsportstut

This adds the DynSports application database that you used in the tutorial as a service in the ICFWS session. It also creates the association between the logical and physical service code names.

- 14 ♦ Save the completed **Service** tab:



- 15 ♦ Exit from the maintenance and control dialog boxes.
- 16 ♦ Verify that the service has been added to the ICFWS session type.

You can verify the settings in the ICWS session type by following the procedure described in the [“Viewing the default ICFWS session settings”](#) section.

Basically, you run the Session Type Maintenance tool and check the Session Services node under ICFWS. After completing the tasks described in this section, the Session Services list should show an entry for dynsportstut:



To add a new Repository database, you would perform the same procedure described in this section. However, you would also remove the default Repository session service, which is described in the [“Removing a session service from ICFWS”](#) section.

Removing a session service from ICFWS

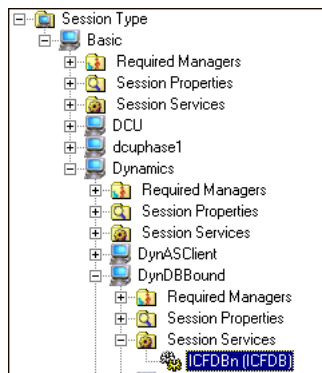
The installation of Progress Dynamics creates an ICFWS session with a network connection to the default Repository (ICFDBn). If you create a separate Repository for your application (as you do in the [Getting Started with Progress Dynamics](#) tutorial), you should add the application Repository as a session service and remove the default Repository as a session service.

Follow the procedure described in the [“Adding a database to ICFWS as a session service”](#) section to add the application Repository to ICFWS. Then, to remove the default Repository service, follow these steps:

- 1 ♦ Start a Progress Dynamics session that has administrative privileges.

For the tutorial application, use the shortcut that you used to start a development session for the tutorial.

- 2 ♦ From the Progress Dynamics AppBuilder **Tools** menu, select **Administration**.
- 3 ♦ From the Administration **Session** menu, select **Session Type Control**.
- 4 ♦ In the **Session type code** lookup field at the top left of the window, enter **DynDBBound** and click **Find** to expand the session type tree.
- 5 ♦ Expand the **DynDBBound** node, and expand the **Session Services** node below it.
- 6 ♦ Select the session service that you want to remove. The following shows the **ICFDBn** service selected:



- 7 ♦ Choose **Delete Record** from the **File** menu, or click  on the Session Service Maintenance toolbar in the right frame.

A confirmation window appears. Click **OK** confirm the deletion.

NOTE: This procedure does not completely remove ICFDBn. It removes it from ICFWS and any other session types that inherit from DynDBBound as a session service, but ICFWS remains available to add to other session types.

3.1.3 Creating a new ICFWS session

Instead of modifying a session type, you can create a new session type that is tailored for your application. You do this by either extending or copying an existing session type. The new session type inherits from its parent type all managers, session properties, and session services, which you can override if necessary; you can also add new ones as appropriate.

This section explains how to extend the default ICFWS to create a session type for the tutorial application. See the [Progress Dynamics Administration Guide](#) for detailed information on extending and copying session types.

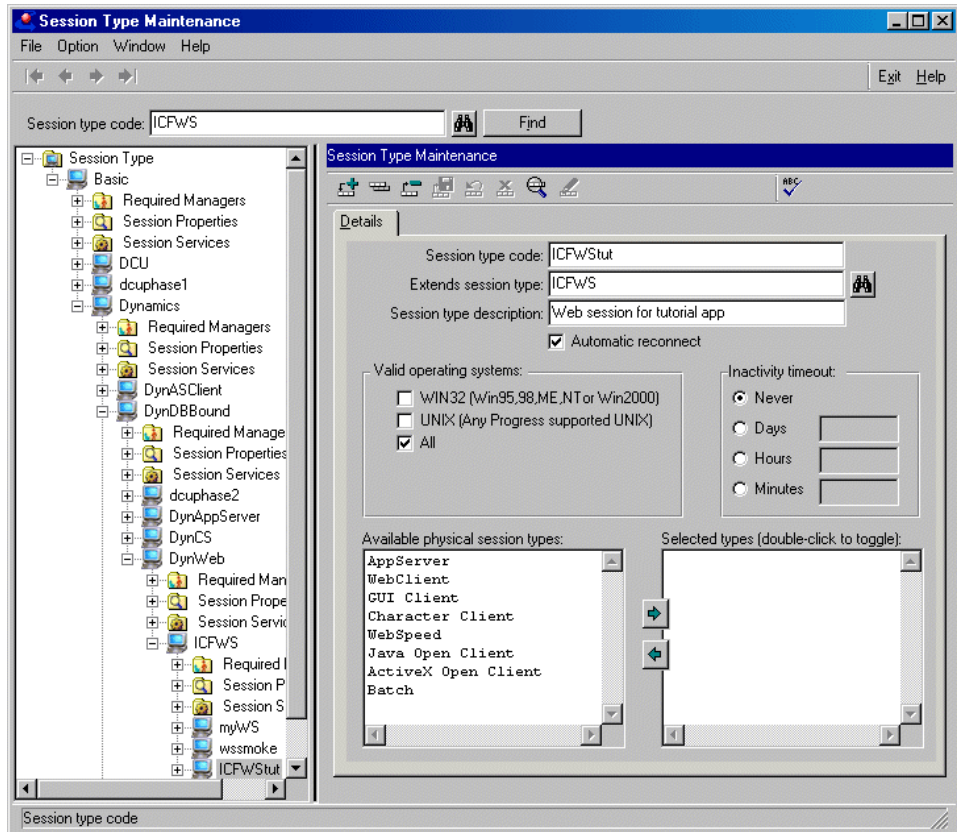
From the Session Type Maintenance tool:

- 1 ♦ Start a Progress Dynamics session that has administrative privileges. For the tutorial application, use the shortcut that you created to start a development session for the tutorial.
- 2 ♦ From the Progress Dynamics AppBuilder **Tools** menu, select **Administration**.
- 3 ♦ From the Administration **Session** menu, select **Session Type Control**.
- 4 ♦ In the **Session type code** lookup field at the top left of the window, enter **ICFWS** and click **Find**.
- 5 ♦ Right-click the **ICFWS** node and click **Add Session Type** on the context menu. The Session Type Maintenance window for the new type appears in the right frame.

- 6 ♦ Enter a unique **Session type code** and **Session type description** that will allow you to quickly identify the session type. For example, you could name the ICFWS session for the tutorial application **ICFWStut** and describe it as **Web session for tutorial app**.

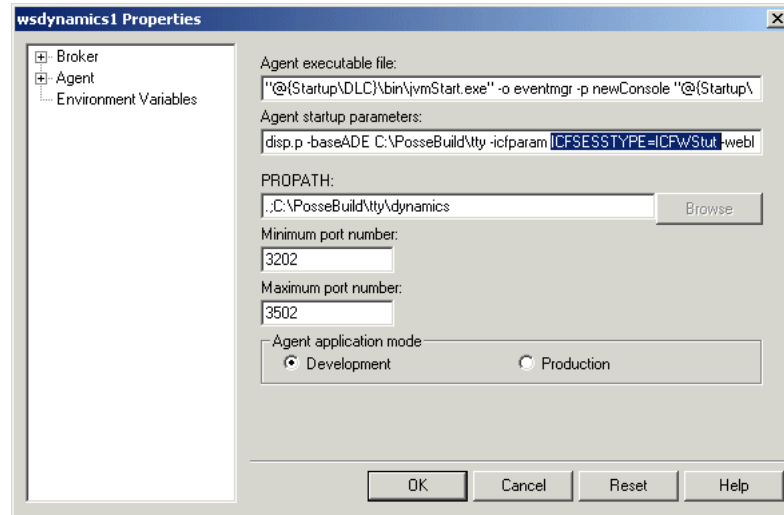
It is not necessary to change any of the other settings.

- 7 ♦ Choose the **Save** button on the toolbar. The new session type appears in the tree view as a child of the ICFWS type:



- 8 ♦ Using Progress Explorer, change the session type in the startup parameters of the Progress Dynamics WebSpeed Agent.

By default, Progress Dynamics creates a WebSpeed Broker, `wsdynamics1`, with the session type set to ICFWS. Change the session type of the Agent by resetting the `-icfparam ICFSESTYPE=ICFWStut` startup parameter. For example:



3.1.4 Updating the XML configuration file

The configuration file, `icfconfig.xml`, defines each session type so that the information is available to the Configuration File Manager at startup. The definition of each session type includes a list of required managers, session properties, and session services.

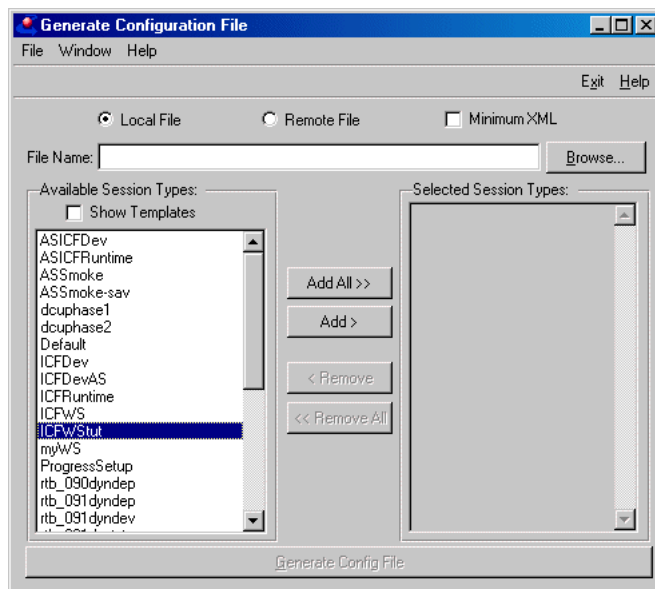
When you create an application, like the tutorial, you usually create an `icfconfig.xml` in the application's work directory. When you modify a session type (like ICFWS) or create a new session type (like ICFWStut), you must update the application's `icfconfig.xml` file.

To generate (or update) your XML configuration file, follow these steps:

- 1 ♦ Start a Progress Dynamics session that has administrative privileges.

For the tutorial application, use the shortcut that you used to start a development session for the tutorial.

- 2 ♦ From the Progress Dynamics AppBuilder **Tools** menu, select **Administration**.
- 3 ♦ From the Administration tool, choose **Session**→**Generate Configuration File** to display a window like the following:



- 4 ♦ In the list of available session types in the left pane, select one or more entries and click **Add**. The selected types appear in the right pane.

You can include configuration information for multiple session types in a single file, but for this example, just select the session type that you created for the tutorial application.

- 5 ♦ Before you generate your configuration file, click the **Browse** button and select the appropriate directory as the target location for the file. The default `icfconfig.xml` file is installed in `install_dir\gui\icf`. Since you do not want to overwrite the default, you should generate an XML configuration file in your application's work directory or some other location.

You might also want to give the XML file an identifiable name, `icfconfigtut.xml` for example. One advantage of specifying a different name is that you avoid confusion with the default XML configuration file. However, be sure that the filename has a `.xml` extension.

- 6 ♦ Click the **Generate Config File** button. The tool generates definitions for the selected session types and writes the information to the configuration file. A confirmation message appears when the generation is successful.
- 7 ♦ Choose the **Close** button to close the Generate Configuration XML file dialog box.
- 8 ♦ Close the Session Type Control window.
- 9 ♦ Close the Dynamics AppBuilder to exit the current Progress Dynamics session.
- 10 ♦ Stop and restart the Progress Dynamics DB servers.

For more information about XML configuration files, see the [Progress Dynamics Administration Guide](#).

3.2 Configuring your Web server

Basically, you configure your Web server so that it can find the Progress Dynamics static files and the appropriate WebSpeed Broker for Progress Dynamics. This section provides general information about both tasks.

It also provides specific information about configuring the Microsoft IIS and the Apache Web servers. If you are running a Web server other than IIS or Apache, you must refer to the documentation for that Web server for configuration information.

3.2.1 Specifying the location of static files

Static files include HTML, JavaScript, Cascading Style Sheet, application, and graphics files. The Web server can find these files through the use of virtual directories, or it can find them when they are installed under the Web Server's Document Root directory.

You choose where to put the static files during the installation of Progress Dynamics. [Figure 3-2](#) shows the Progress Dynamics installation dialog where you can choose to copy the files to the Document Root Directory. The actual location of the Document Root Directory depends on your Web server. You specify the location of the Document Root Directory when you install WebSpeed.

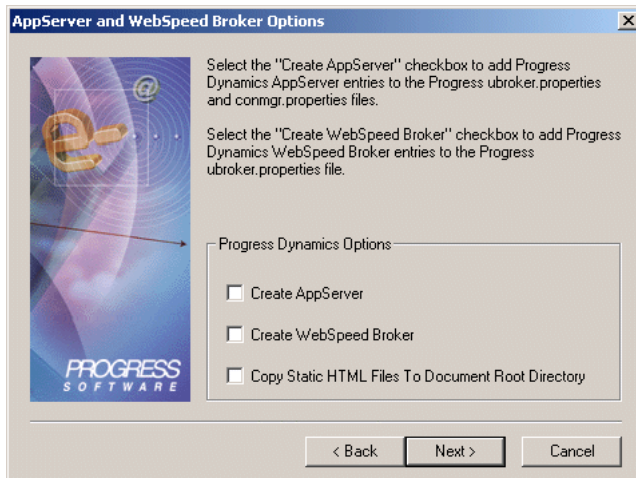


Figure 3-2: AppServer and WebSpeed Broker Options

Rather than physically moving static files into or below the Document Root, you can set up a virtual directory that points to the directory (*install_dir\http\conf\ry*) that contains the static files. For more information on setting up virtual directories, see the [“Configuring the IIS Web server”](#) section or the [“Configuring the Apache Web server”](#) section. Refer to individual product documentation for other Web servers.

The static files for Web applications reside under a relative path structure with main module files separated into subdirectories, as shown in [Table 3–2](#).

Table 3–2: Directory structure for static files

Directory	Contents
[<i>path</i>]	Link pages for testing and tools (index.html, default.htm)
[<i>path</i>]\dhtml	DHTML files: .css, .htm, and .js
[<i>path</i>]\img	Images

NOTE: In table [Table 3–2](#), *path* can be either a virtual directory that points to *install_dir\http\icf\ry* or it can be *document_root_dir\ry* (where *document_root_dir* is the Document Root directory of your Web server).

3.2.2 Specifying the appropriate Progress Dynamics Broker

For standard WebSpeed applications, you specify a Broker in a URL (for example, `WService=wsbroker1`). However, for Progress Dynamics Web applications, the Broker is either specified in the default HTML layout file (`default.htm`), or through extension mapping on the Web server.

Specifying a Broker in the default HTML layout file

If you want to specify your Broker in the default HTML file (`default.htm`), you must set the `dyn` attribute to point to a Progress Dynamics Broker.

The markup should look similar to the bold text in the following:

```
<td id="app" dyn="/scripts/cgiip.exe/WService=wsdynamics1/dhtml"
css="../dhtml/ryapp.css" timeout="20000" encoding="UTF-8" lognote="on">
```

You can have multiple Brokers running for a single Web server. However, a default HTML layout page can only be associated with one named Broker. If you have multiple Brokers, you must create a unique default HTML page for each Broker.

Specifying a Broker using extension mapping

Extension mapping associates the `.icf` extension with a named Progress Dynamics Broker. The `.icf` extension is generated when the Web server accesses `default.htm` in the `/dhtml` directory.

To enable extension mapping:

- 1 ♦ Verify that the `dyn` attribute in the default HTML layout file is **not** set to any value. (No value is the default setting.) The markup should look similar to the bold text in the following:

```
<td id="app" dyn="" css="../dhtml/ryapp.css" timeout="20000"
encoding="UTF-8" lognote="on">
```

NOTE: A Broker specified in the `dyn` attribute overrides a Broker specified through extension mapping.

- 2 ♦ Configure your Web server. For specific information, see the [“Configuring the IIS Web server”](#) section or the [“Configuring the Apache Web server”](#) section. Refer to individual product documentation for other Web servers.

When you use extension mapping, only one Progress Dynamics Broker can be set up for the Web server.

3.2.3 Configuring the IIS Web server

These configuration instructions apply to the Microsoft IIS Web server. Configuration tasks include:

- [Setting up virtual directories](#)
- [Setting up extension mapping](#)

Setting up virtual directories

You create a virtual directory for Progress Dynamics static files so that the Web server can find them. (The other alternative is to copy the static files to the Web server’s Document Root directory.) You can think of a virtual directory as an alias for `install_dir\tty\icf\ry`, the directory where the Progress Dynamics install places the static files.

You can use the IIS Web server's Internet Service Manager to set up virtual directories. The Internet Service Manager can usually be found under the Administrative Tools section of the Windows Control Panel. When it is running, select **Default Web Site**. Then select **Action**→**New**→**Virtual Directories**. A wizard takes you through the process of creating a virtual directory.

Setting up extension mapping

You use extension mapping to enable the Web Server to find the Progress Dynamics WebSpeed Broker. (The other alternative is to explicitly name the Broker in the default HTML layout file.)

This section contains instructions on setting up extension mapping for `.wsc` and `.icf` on the IIS Web server.

NOTE: Extension mapping for `.wsc` (in addition to `.icf`) is necessary to enable the WebSpeed CGIIP Messenger to run under IIS. For more information, read the `cgiip.wsc` file. It is located in `install_dir\bin`, where `install_dir` is the folder where Progress Version 9.1D is installed.

To set up extension mapping, follow these steps:

- 1 ♦ Start the Web server if it is not already running.
- 2 ♦ Start the Internet Services Manager.

You can usually find the Internet Services Manager under **Administrative Tools** in the Windows Control Panel.

- 3 ♦ Under the **Web Sites** node, expand the **Default Website** node.
- 4 ♦ Right-click on the **Scripts** folder.

The **Scripts** folder should be a virtual directory that points the Web server's scripts directory, usually `C:\Inetpub\Scripts`. If it does not exist, you can create it. See the [“Setting up virtual directories”](#) section.

- 5 ♦ Select **Properties** from the drop-down menu.

- 6 ♦ In the **Virtual Directory** tab, choose **Configuration**.
- 7 ♦ In the **App Mappings** tab, choose **Add**.
- 8 ♦ In the **Add/Edit Application Extension Mapping** pop-up window, enter the following in the **Executable** field:

`install_dir\bin\cgiip.exe %s %s`

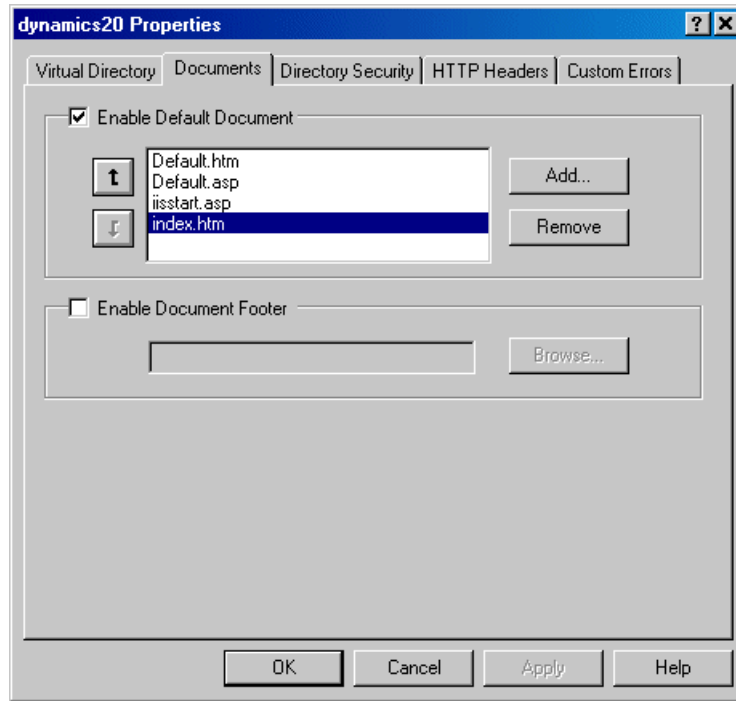
where *install_dir* is the directory where you installed Progress Version 9.1D. (If *install_dir* contains spaces, you must add quotes around it.)

- 9 ♦ Enter the following in the **Extension** field:

`.WSC`

- 10 ♦ Select the **Script Engine** check box. Deselect the **Check that file exists** check box.
- 11 ♦ Choose **OK**.
- 12 ♦ In the **App Mappings** tab, choose **Apply**. Then choose the **OK** button.
- 13 ♦ Right-click on the virtual directory that was added for Progress Dynamics (for example, **dynamics20**) under the **Default Website** node.
- 14 ♦ Select **Properties** from the drop-down menu.

- 15 ♦ In the **Documents** tab, check the **Enable Default Document** option. The settings should look similar to the following:



- 16 ♦ In the **Virtual Directory** tab, choose **Configuration**.
- 17 ♦ In the **App Mappings** tab, choose **Add**.
- 18 ♦ In the **Add/Edit Application Extension Mapping** pop-up window, enter the following in the **Executable** field:

```
install_dir\bin\cgiip.exe -i wsdynamics1
```

where *install_dir* is the directory where you installed Progress Version 9.1D. (If *install_dir* contains spaces, you must add quotes around it.) *wsdynamics1* is the name of the Progress Dynamics Web Broker.

- 19 ♦ Enter the following in the **Extension** field:

```
.icf
```

20 ♦ Select the **Script Engine** check box. Deselect the **Check that file exists** check box.

21 ♦ Choose **OK**.

22 ♦ In the **App Mappings** tab, choose **Apply**, and then choose **OK**.

23 ♦ Exit from the Internet Service Manager.

NOTE: If you use extension mapping, only one Progress Dynamics Broker can be set up for the Web server.

3.3 Configuring the Apache Web server

These configuration instructions apply to the Apache Web Server Version 1.3 and later. Configuration tasks include:

- [Setting up virtual directories](#)
- [Setting up extension mapping](#)

The Apache Web Server must be restarted after you make configuration changes.

Setting up virtual directories

You create a virtual directory for Progress Dynamics static files so that the Web server can find them. (The other alternative is to copy the static files to the Web server's Document Root directory.) You can think of a virtual directory as an alias for *install_dir\tty\icf\ry*, the directory where the Progress Dynamics install places the static files.

For the Apache Web server, edit (*apache_install_dir/conf/httpd.conf*) and create a *dynamics21* alias for *dynamics_install_dir/tty/icf/ry*. The following example shows the entry:

```
Alias /dynamics21/ "dynamics_install_dir/tty/icf/ry/"
<Directory "dynamics_install_dir/tty/icf/ry">
    Options Indexes MultiViews
    AllowOverride None
    Order allow,deny
    Allow from all
</Directory>
```

Setting up extension mapping

You use extension mapping to enable the Web Server to find the Progress Dynamics WebSpeed Broker. (The other alternative is to explicitly name the Broker in the default HTML layout file.)

For the Apache Web server, edit (*apache_install_dir/conf/httpd.conf*) to add a directive (*ScriptAliasMatch*) that allows for mapping of the *.icf* extension. The first two lines in the following example shows the entry (in the actual file the entry must be on a single line):

```
ScriptAliasMatch (.*)\.icf$
"apache_install_dir/cgi-bin/cgiip.exe/WSservice=wsdynamics1/$1"
Alias /dynamics21/ "dynamics_install_dir/tty/icf/ry/"
<Directory "dynamics_install_dir/tty/icf/ry">
    Options Indexes MultiViews
    AllowOverride None
    Order allow,deny
    Allow from all
</Directory>
```

The order of directives in the Apache configuration file is important. Be sure that the *ScriptAliasMatch* directive preceeds the *Alias* and *Directory* directives, as shown above.

NOTE: If you use extension mapping, only one Progress Dynamics Broker can be set up for the Web server.

3.4 Configuring the WebSpeed Broker and Agents

Before configuring your WebSpeed Broker and Agents, you should verify your session setup. See the [“The ICFWS session”](#) section. Also, you should configure the ICFWS session so that the Progress Dynamics Agents can connect to an application database. See the [“Adding a database to ICFWS as a session service”](#) section.

To develop or deploy Progress Dynamics browser-based applications, you must set up a Progress Dynamics version of the WebSpeed Broker and Agents. The Dynamics installation creates a default Broker (called *wsdynamics1*) and Agents. You can use the default Broker or you can create your own, as described in this section.

NOTE: Although Progress Dynamics uses most of the standard WebSpeed components, the Agent is quite different from the standard WebSpeed Agent. The Progress Dynamics version of the WebSpeed Agent is actually comprised of managers that interact with the Repository and with the application database. For more information, see [Chapter 2, “Progress Dynamics Web Architecture.”](#)

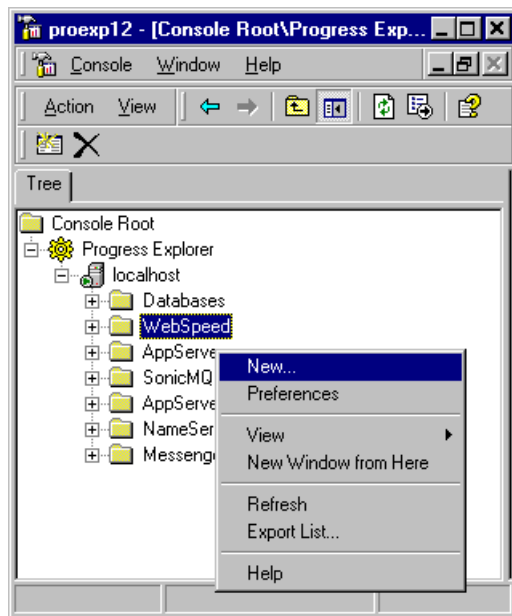
The following procedure describes how to configure and start the Progress Dynamics Broker and Agents using the Progress Explorer tool. If you prefer (or if you are working from a UNIX system), you can accomplish the same task by manually editing your `ubroker.properties` file. For more information about Progress Explorer and about editing the `ubroker.properties` file, see the *Progress Installation and Configuration Guide Version 9 for Windows*.

To configure and start the Progress Dynamics Broker, follow these steps:

- 1 ♦ Make sure that your Progress Version 9.1D AdminServer is running. (Progress Explorer requires a running AdminServer.)
- 2 ♦ Start Progress Explorer.

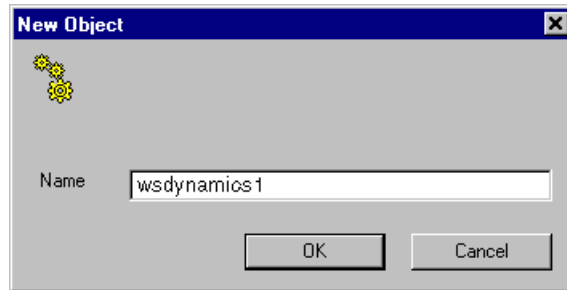
Select **Progress Explorer Tool** from the Progress Version 9.1D folder on your **Start** menu.

- 3 ♦ Connect Progress Explorer to the local host or to the machine that will run the Progress Dynamics Broker.
- 4 ♦ Right-click on the **WebSpeed** node and select **New** from the pop-up menu, as shown in the following figure:

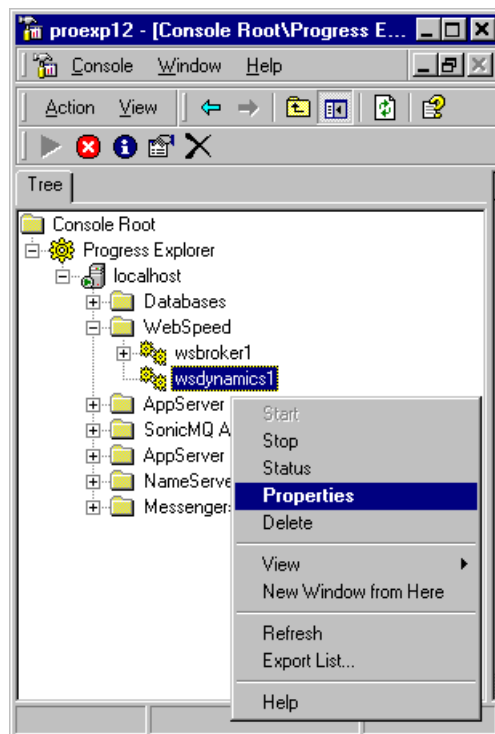


The **New Object** dialog box appears.

- 5 ♦ Type a name for your Broker, then choose **OK**. The following figure shows an entry for a new Broker named **wsdynamics1**:



- 6 ♦ Right-click on the new Progress Dynamics Broker name in the tree view and select **Properties** from the pop-up menu. The following figure shows the selection of a Broker named **wsdynamics1**:



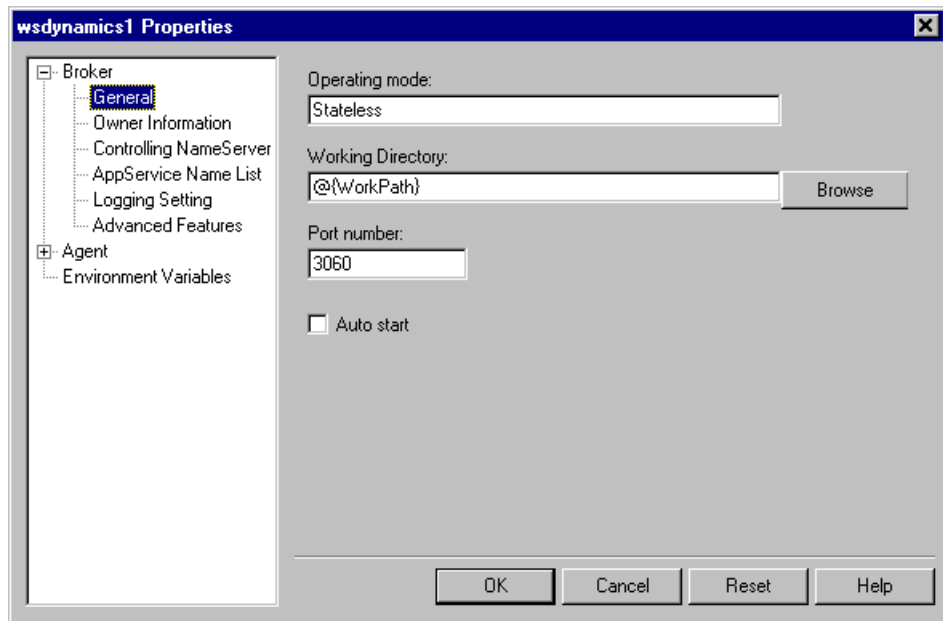
NOTE: The following steps indicate which property settings that you might need to change. You can accept the defaults for properties that are not mentioned.

- 7 ♦ Select the **General** node under **Broker**. Set the working directory and port number, then, choose **OK**.

The **Working Directory** text box contains a variable (@{WorkPath}) that points to the work directory that you specified during Progress Version 9.1D installation. Change this if you are using a different directory for Progress Dynamics work.

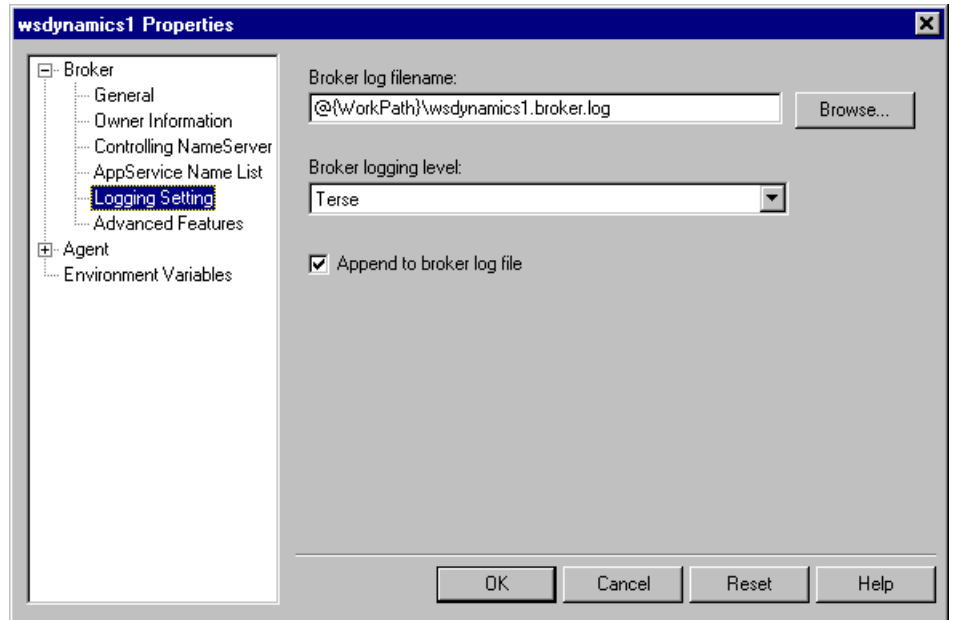
Enter any port number that is not in use on your system.

The following figure shows an example of a completed **Broker Properties** dialog box:



- 8 ♦ Select the **Logging Setting** node under **Broker**. Type the path for your Broker's log file, then choose **OK**.

The following figure shows a log file specified at the top level of the work directory:



NOTE: If you are debugging Broker problems, change the logging level from **Terse** to **Verbose**. However, change the logging level back to **Terse** after you resolve the problems. The **Verbose** mode creates very large log files.

- 9 ♦ Select the **General** node under **Agent**. Edit the **Agent startup parameter** and **PROPATH** fields.

You must specify the ICFWS Session Type by adding `-icfparam ICFSESSTYPE=ICFWS` to the **Agent startup parameters** field.

You can also create a parameter (`.pf`) file, which is usually located in the top level of your installation directory. Use the following syntax to specify a parameter file in the **Agent startup parameters** field:

```
-pf filename.pf
```

The following example shows the contents of a typical parameter file for a Progress Dynamics Agent:

```
/* wsdynamics.pf */
-p          web\objects\web-disp.p
-inp        18000                # max-statement-length
-mmax       65534                # max-r-code-memory
-tok        1600                # max-tokens
-icfparam   ICFSESSTYPE=ICFWS,ICFCONFIG=install_dir\tty\icfconfig.xml
-debugalert
-weblogerror
```

NOTE: Replace *install_dir* with the pathname of your Progress Dynamics installation directory.

You must precede the default PROPATH setting (@{WinChar Startup\PROPATH}) with entries similar to the following example:

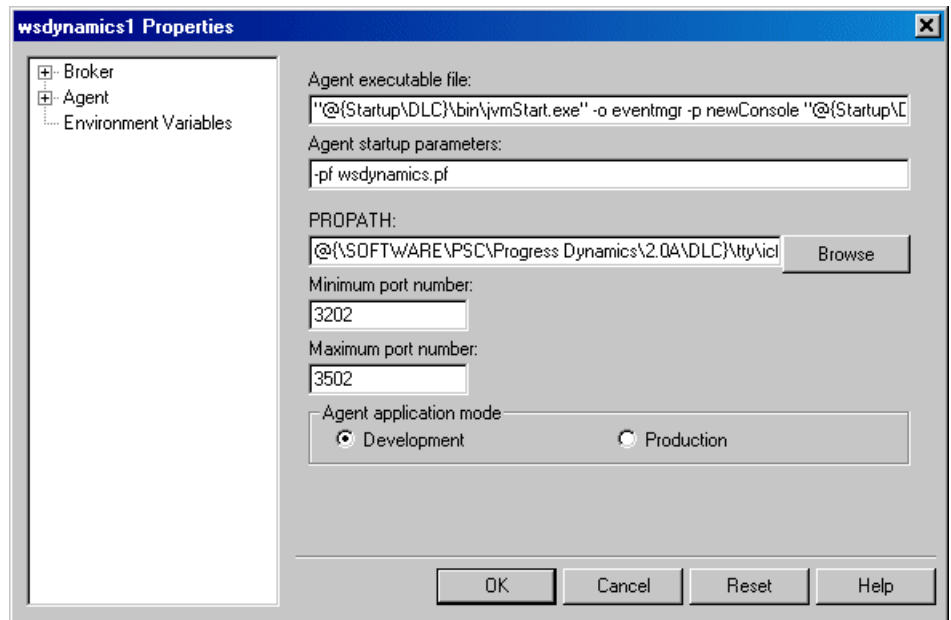
```
. ;
@{\SOFTWARE\PSC\Progress Dynamics\2.1A\DLC}\tty\dynamics;
@{\SOFTWARE\PSC\Progress Dynamics\2.1A\DLC}\tty;
@{\SOFTWARE\PSC\Progress Dynamics\2.1A\DLC};
@{\SOFTWARE\PSC\Progress Dynamics\2.1A\DLC}\src;
```

You enter PROPATH settings on a single line. The settings in the example above are separated by returns only because a single line would not fit on the page.

NOTE: These PROPATH settings use a registry key (@{\SOFTWARE\PSC\Progress Dynamics\2.1A\DLC}) to form the first part of the pathname. Instead of using the registry key syntax, you can specify the actual pathnames (for example, C:\Program Files\Progress Dynamics\tty).

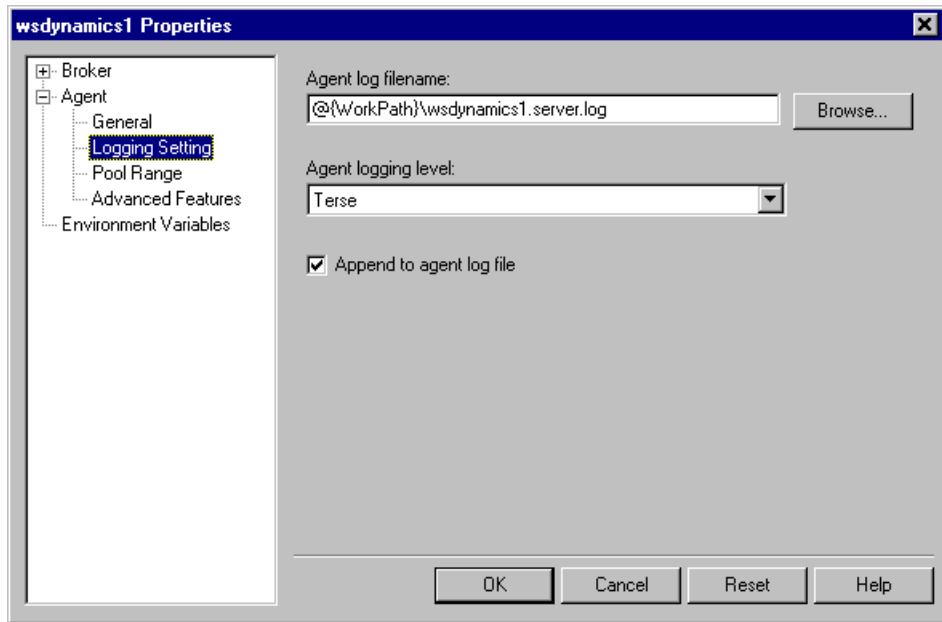
- 10 ♦ Set **Agent application mode** to **Development** if the Agent runs in a development environment, or to **Production** if the Agent runs in a deployment environment.

The following figure is an example of general settings for a Progress Dynamics Agent:



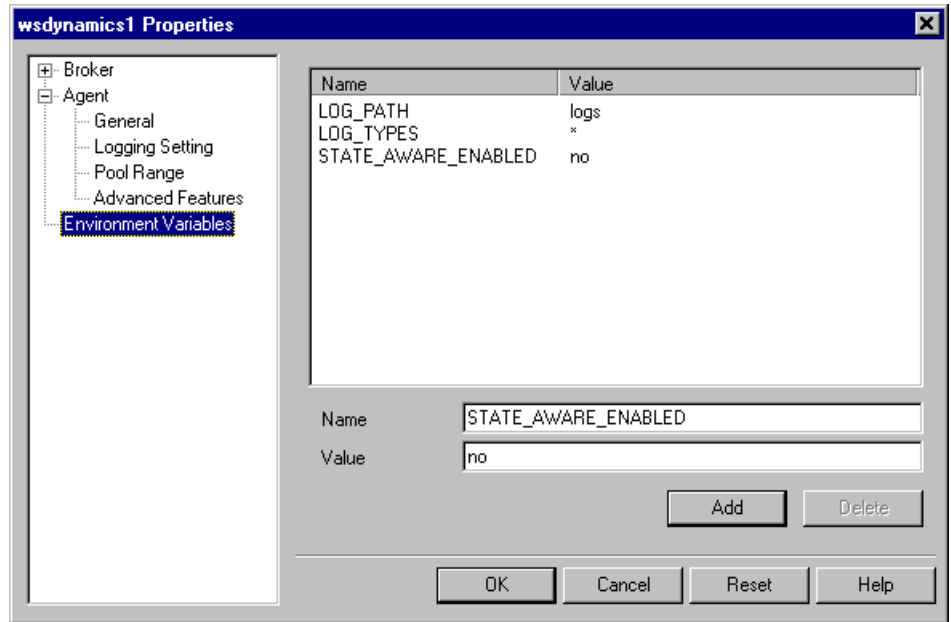
- 11 ♦ Choose **OK**.

- 12 ♦ Select the **Logging Setting** node under **Agent**. Enter the path for your Agents's log file, then choose **OK**. The following figure shows a log file specified at the top level of the work directory:



NOTE: If you are debugging Agent problems, change the logging level from **Terse** to **Verbose**. However, change the logging level back to **Terse** after you resolve the problems. The **Verbose** mode creates very large log files.

- 13 ♦ Select the **Environment Variables** node and enable the appropriate variables. The following figure shows some common settings:



In the preceding example:

- LOG_PATH specifies a directory (either a full pathname or a path relative to your Progress work directory) where log files will be written. Note that the Broker will not create the logging directory. You must create it if it does not already exist.
- LOG_TYPES specifies what types of messages to log. In this case, the asterisk (*) indicates that all messages will be logged.
- STATE_AWARE_ENABLED specifies whether or not to load state-aware enabling code into memory. The default is yes. However, you can improve performance if all of your Web applications are stateless (which is the case for Progress Dynamics Web applications) by setting this variable to no.

NOTE: STATE_AWARE_ENABLED does not automatically make Web applications run in the state-aware mode. It merely enables Agents to handle applications that are already written to be state-aware.

For more information on environment variables, choose the **Help** button in the **Environment Variables Properties** dialog box. The online help describes how to create and use variables. It also contains a list of the predefined variables and describes their use.

- 14 ♦ Start your Progress Dynamics Broker.

You can start the Broker with Progress Explorer or with the `wtbman` command. See the *WebSpeed Installation and Configuration Guide* for more information.

NOTE: Depending on which Progress Version 9.1D products you have installed on your system, you might exceed the number of database connections allowed by your license. If you exceed the number of database connections allowed by your license, you will see a database connection error message when you try to start the Broker

To bypass this situation, reduce the number of Agents started by the Broker. Also, shut down any databases that are not being used.

For more information about database connection limitations, see the *Progress Installation and Configuration Guide Version 9*. For more information about reducing the number of running Agents, see the *WebSpeed Installation and Configuration Guide*.

3.5 Testing the Web server

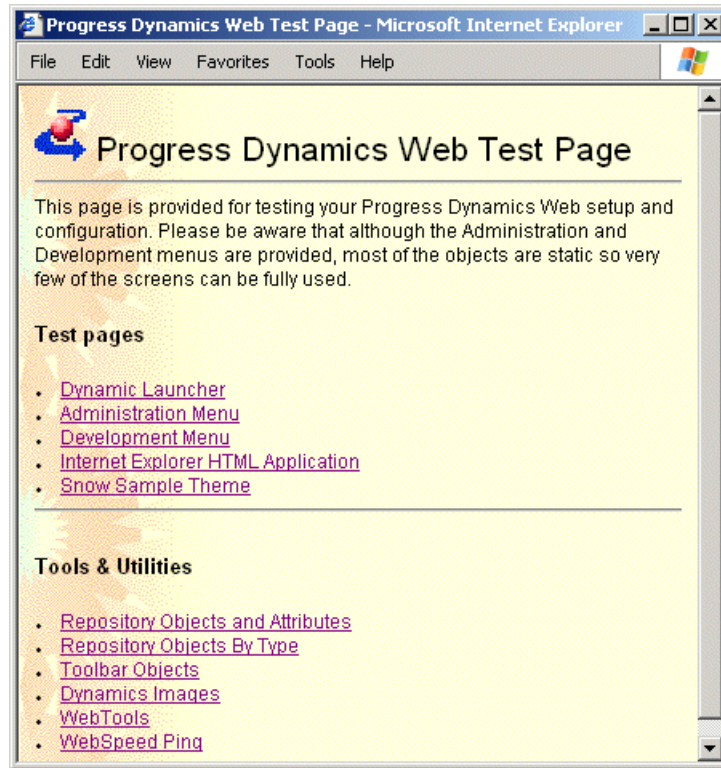
Before you can test the Web server, you must have a Progress Dynamics Broker. If you have not set up a Broker, see the [“Starting an application development session”](#) section.

To verify that your Web server has been set up correctly, follow these steps:

- 1 ♦ Verify that the Web server is running.
- 2 ♦ Start the Internet Explorer browser. (Version 6.0 is recommended.)
- 3 ♦ Enter the name of the machine where the Web server is running followed by the virtual directory name (or the appropriate Document Root directory) in the **Address** field.

For example: `http://localhost/dynamics21`.

The following page appears:



- 4 ♦ Choose each one of the links and verify that they invoke the appropriate pages.

3.6 Testing the ICFWS session configuration

Before you can test the session configuration, you must have a Progress Dynamics Broker running. If you have not set up a Broker, see the [“Configuring the WebSpeed Broker and Agents”](#) section.

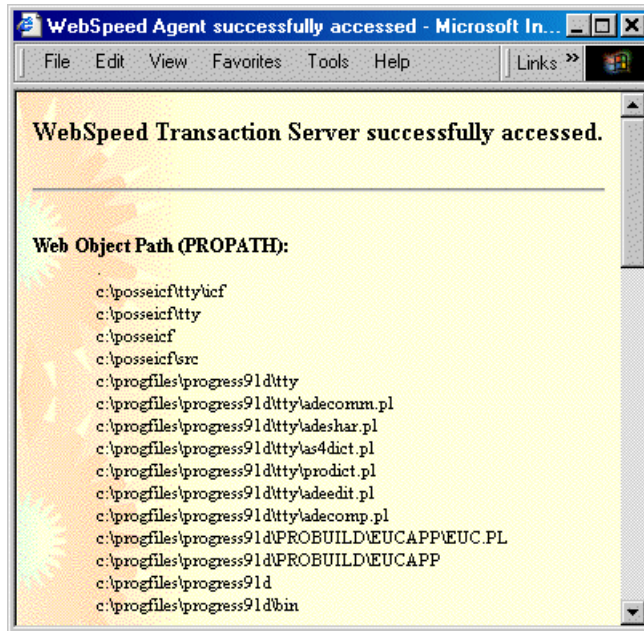
To verify that your ICFWS session is set up correctly, follow these steps:

- 1 ♦ Verify that your Web server is running.
- 2 ♦ Start the Internet Explorer browser. (Version 6.0 is recommended.)
- 3 ♦ Ping the Progress Dynamics Broker, using standard WebSpeed URL syntax.

In standard WebSpeed URL syntax, you specify the path to the appropriate Messenger, the name of the specific Broker, and the command. For example:

```
http://localhost/scripts/cgiip.exe/WService=wsdynamics1/webutil/ping
```

If the ping is successful, you should see a page similar to the following:



NOTE: The pathnames you see on your own system will probably not match the pathnames shown above. Your pathnames will begin with the installation directories for Progress Dynamics and Progress Version 9.1D.

- 4 ♦ Scroll to the bottom of this page and verify that all links and images display correctly.
- 5 ♦ Scroll to the **Checking WebTools Installation** section of the Ping page and select the **here** link.

- 6 ♦ Select **Object State** from the WebTools link list. Verify that the Web objects (managers) in the following table are running:

Manager	Web object
Connection Manager	afconmgrp.r
Referential Integrity Manager	ryrisrvrp.r
Session Manager	afsessrvrp.r
General Manager	afgensrvrp.r
Repository Manager	ryrepsrvrp.r
Security Manager	afsecsrvrp.r
Profile Manager	afprosrvrp.r
Localization Manager	aftrnsrvrp.r
Request Manager	ryreqsrvrp.r
User Interface Manager	ryuimsrvrp.r
Customization Manager	rycusssrvrp.r

- 7 ♦ Select **Databases** from the WebTools link list. Verify that the ICFDB and your application database are connected.

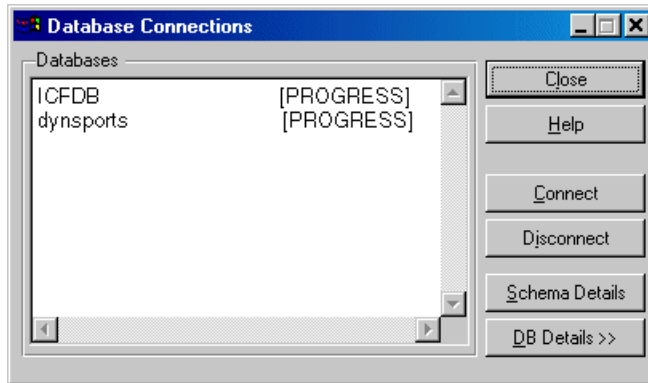
3.7 Starting an application development session

The following steps summarize how to start a development session. For more information, see the [Progress Dynamics Developer's Guide](#).

To start a development session, follow these steps:

- 1 ♦ From the Windows Start menu, select **Start Dynamics DB Servers**.
- 2 ♦ From the Windows Start menu, select **Dynamics Development**.
- 3 ♦ Log in. The Progress Dynamics AppBuilder, the ProTools Palette, and the Object Tools Palette start.

- 4 ♦ From the AppBuilder **Tools** menu, select **Database Connections**. Verify that both your application database and the application Repository (ICFDB database) are connected:



- 5 ♦ From the AppBuilder **Options** menu, select **Preferences**.
- 6 ♦ From the **Preferences** dialog box, select the **WebSpeed** tab.
- 7 ♦ In the **Web Browser** field, enter the pathname of Internet Explorer.

In the **Broker URL** field, enter the URL of the Progress Dynamics Broker. Use the standard WebSpeed URL syntax. For example:

`http://localhost/scripts/cgiip.exe/WService=wsdynamics1`

- 8 ♦ Choose **Test** to verify that your settings are correct.
- 9 ♦ Choose **Save Settings**, then choose **OK**.

Progress Dynamics Web Applications

This chapter contains information on how to create, run, and deploy Progress Dynamics Web applications. It includes the following sections:

- [Overview of Web application development](#)
- [Running a Progress Dynamics Web application](#)
- [URL syntax for Progress Dynamics Web applications](#)
- [Considerations for development and use of Progress Dynamics Web applications](#)
- [Deployment](#)

4.1 Overview of Web application development

To create a Progress Dynamics Web application, you perform the same procedures that you perform when creating a Progress Dynamics Windows application. The general steps are:

1. Create a new product and modules to organize the objects in your application.
2. Import tables from your application database as entities in the Repository.
3. Generate application objects (SDOs, browsers, and viewers).
4. View and edit your application objects.
5. Use the Container Builder to construct a page to hold your application objects.

See the *[Progress Dynamics Developer's Guide](#)* for a detailed description of application development. Also see *[Getting Started with Progress Dynamics](#)* for a tutorial that takes you through application development step by step.

After general application development is complete, you do the following to complete application development for the Web:

1. Set up Progress Dynamics, WebSpeed, and a Web server to run your application, which is described in [Chapter 3, “Setting Up Progress Dynamics for Running Web Applications.”](#)
2. View your application in a browser, which is described in the [“Running a Progress Dynamics Web application”](#) section.
3. Customize your application for the Web, using HTML, Cascading Style Sheets, and JavaScript, which is described in [Chapter 5, “Customizing with Dynamic HTML.”](#)

4.2 Running a Progress Dynamics Web application

Once you have configured Progress Dynamics, WebSpeed, and a Web server, running a Progress Dynamics application in a browser is as simple as supplying the name of a Dynamics container object in a URL query string.

For example, try running the oeCustBrowseWin container. The oeCustBrowseWin container is one of the objects you develop in the tutorial in [Getting Started with Progress Dynamics](#). When you run it using the Dynamic Launcher, the Windows GUI application would look similar to [Figure 4-1](#).

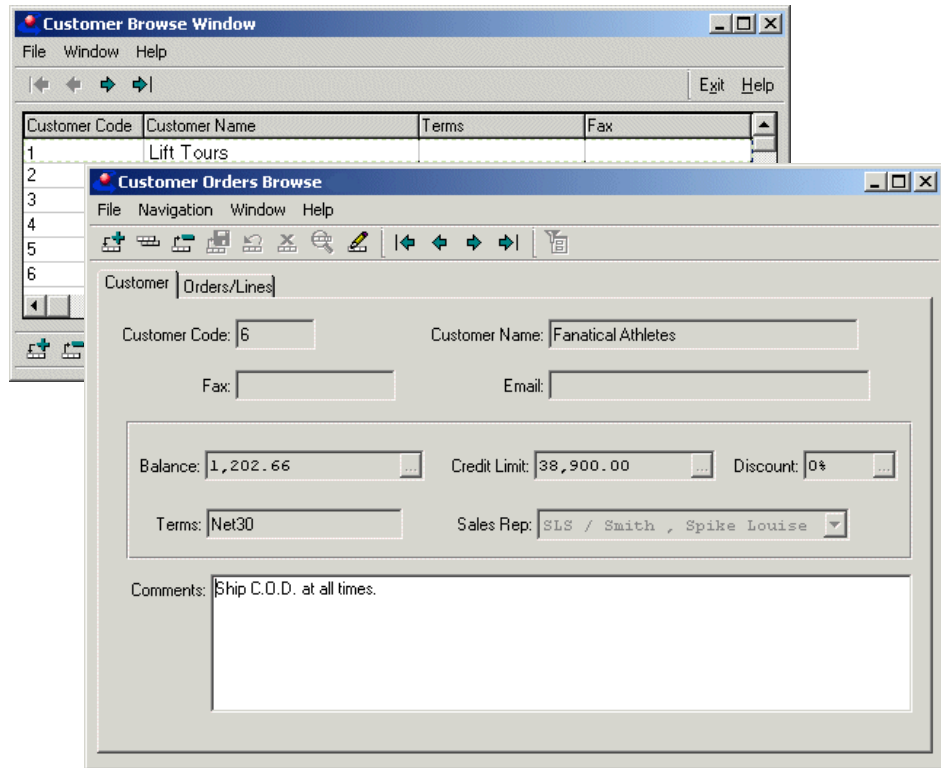


Figure 4-1: An example of a Windows GUI application (oeCustBrowseWin)

The Customer Browse window presents a list of the records in the Customer table of the DynSports sample database. When you click on a record, a second window (Customer Orders Browse) containing two tab folders appears. On the first tab folder, you can view and change customer information. On the second tab folder, you can view and change order information

Figure 4–2 shows the same container object running as a Progress Dynamics Web application.

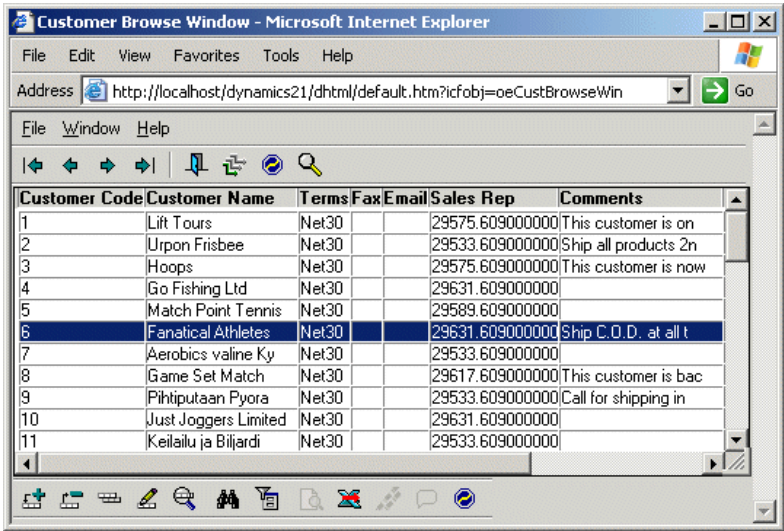


Figure 4–2: An example of a Web application (oeCustBrowseWin)

The application was invoked by entering a URL query string in the browser. Notice that the name of the object, oeCustBrowseWin, is assigned to the icfobj property. For more information about the URL syntax, see the “[URL syntax for Progress Dynamics Web applications](#)” section.

Generally, the look and feel of the Customer Browse running in a Web browser is similar to the same application running in Windows.

When you select a record in the Customer Browse, you see an application for customer and order maintenance. As shown in [Figure 4-3](#), it is similar in look and feel to the same objects running on Windows. However, Customer Orders Browse does not appear in a separate window. It replaces Customer Browse in the same browser window. Also note that you need to use **File** → **Exit** in the application's menu bar to get back to Customer Browse. (The Web browser's **Back** button will not return to Customer Browse. It will return to some previously visited Web page.)

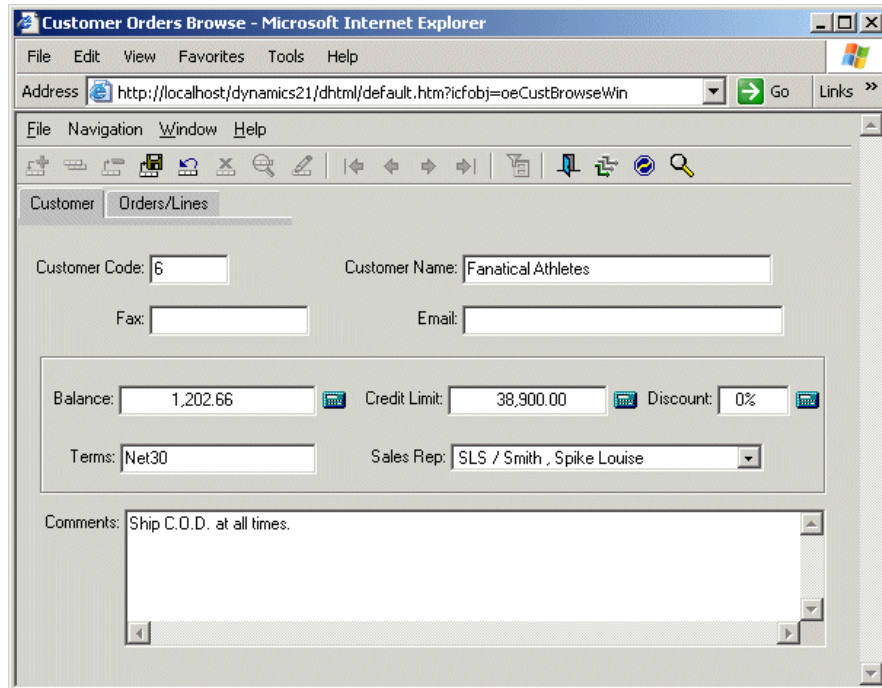


Figure 4-3: Customer Orders Browse running in a Web browser

4.3 URL syntax for Progress Dynamics Web applications

The URL syntax for accessing Progress Dynamics Web applications is:

SYNTAX

```
http://host_name[:Port_Num]/{virtual_dir|actual_path}  
/dhtml/default.htm?icfobj=object_name
```

host_name[:Port_Num]

Specifies the name of the machine running the Web server. You might also need to specify a port number if your Web server uses something other than the default port number (which is usually 80).

virtual_dir|actual_path

Specifies the directory where the Progress Dynamics static files are installed. This might be a virtual directory that you configured during installation. You can also specify the actual pathname. However, the advantage of using virtual directories is that you can hide file location details from your end users.

dhtml

Specifies the directory (under *virtual_dir* or *actual_path*) where the static CSS, JavaScript, and HTML files are located.

default.htm

Specifies the default HTML page, in which the Progress Dynamics object runs. There is a sample default HTML layout page (*default.htm*) in the *dhtml* directory. For more information about *default.htm*, see the “Default HTML file” section in [Chapter 5](#), “Customizing with Dynamic HTML.”

NOTE: Unlike standard WebSpeed, the Broker is not directly stated in the URL. The Progress Dynamics Broker is located either through a reference in the default HTML layout page or through extension mapping. For more information see the “Specifying the appropriate Progress Dynamics Broker” section in [Chapter 3](#), “Setting Up Progress Dynamics for Running Web Applications.”

icfobj=object_name

Specifies which Progress Dynamics object to run. The *icfobj* variable is recognized by the Request Manager as containing the name of an object in the Repository.

4.4 Considerations for development and use of Progress Dynamics Web applications

Progress Dynamics (Version 2.x) allows you to develop full-featured Web applications. Full-featured Web applications implement the functionality and user interface features that you find in Windows applications. However, it is not possible to replicate exactly Windows-based applications in a Web environment. Not all Windows features translate well to the Web browser environment because of differences in event models or object toolkits. Moreover, it is sometimes not desirable to translate Windows features to a Web browser because there are alternative visualizations and behaviors that come closer to what users expect in a Web browser environment.

The significant issues are highlighted in the sections that follow:

- [General considerations for Web look and feel](#)
- [Development requirements and restrictions](#)
- [Unsupported features and objects](#)
- [Browser behavior](#)

4.4.1 General considerations for Web look and feel

This section discusses certain guidelines for designing applications in a manner consistent with prevailing Web standards.

Dialog boxes

The following three types of dialog boxes can be replicated in a Web browser:

- A *modal dialog box* is a separate browser window instance that pops up over the main base window and receives focus. You cannot access the base window until the modal dialog box has been closed.
- A *nonmodal dialog box* is actually a separate browser window instance that pops up over the main base window and receives focus. You can access the base window without closing the nonmodal dialog box.
- An *overlay dialog box* replaces the contents of the main browser window frame, so no additional Web browser window instance is created. There is some object (a button or a link, for example) that lets the user return to the main window frame. (The Progress Dynamics framework supports overlay dialog boxes for Find/Filter and field lookups.)

Modal dialog boxes are not very common in a Web browser GUI environment. You are more likely to see overlay dialog boxes. Although nonmodal dialog boxes are frequently implemented as pop-up advertisements in Web sites, they are less common in Web applications. Overlay dialog boxes are usually preferable.

Alternative object visualizations

The following objects have alternative visualizations that change the look and feel of an application to make it more “Web-like” without changing the underlying functionality:

- Tab folders can appear as multi-page wizards with navigation buttons.
- Menu bars can appear as tree views with a hierarchy of collapsible and expandable nodes.

4.4.2 Development requirements and restrictions

This section details specific programming considerations that you should be aware of when developing Progress Dynamics applications for use in a Web environment.

Custom UI events

Custom UI events applied to dynamic viewer fields create dynamic triggers that either run a procedure or publish an event. If these triggers contain code that runs in a Web application, you must reproduce them in JavaScript. For more information on creating and deploying custom behaviors, see [Chapter 5, “Customizing with Dynamic HTML.”](#)

ActiveX controls

Progress Dynamics **does not** support ActiveX controls for dynamic objects. Technically, it may be possible to emulate ActiveX controls through JavaScript. However, Progress Dynamics cannot register ActiveX controls in the repository as dynamic objects. Therefore, ActiveX controls are not dynamically rendered in Progress Dynamics Web applications.

Also note that ActiveX controls can be problematical for Internet applications in general because:

- They require a lower level of security in Web browsers.
- They require downloads that are operating-system-specific.

Client-side logic

Progress Dynamics SmartDataViewers store all UI logic as Progress 4GL procedures running persistently on the 4GL client. When the client is a Web browser, however, you must implement the client-side logic in JavaScript.

Progress Dynamics allows you to attach JavaScript functions for adding, updating, deleting, and validating records to WebDataObjects (WDOs). (WDOs are the client-side equivalents of SDOs). For more information, see [Chapter 5, “Customizing with Dynamic HTML.”](#) In addition, see the “[Client logic](#)” section in [Chapter 6, “Server-side Business Logic and Client Logic.”](#)

I/O blocking in server-side business logic

I/O blocking statements must be eliminated in server-side business logic. Progress Dynamics Web applications do not support user prompts or messages that are generated by business logic executing on the server side.

Menu and button enable/disable rules

Progress Dynamics Web applications implement the standard ADM2 rules for enabling and disabling of menu items and toolbar buttons. For example, menu selections are disabled or “grayed out” in response to application state or user actions.

The Toolbar and Menu Designer tool allows you to modify the enable and disable rules for Windows applications. However, modifications are not supported for Web applications. Web applications support only the default rules.

Server connection ID

For Progress Dynamics Web applications, it is recommended that you disable the generation of SERVER-CONNECTION-ID in the WebSpeed Messenger (cgip.exe).

Dynamic lookups

Developers should be aware of the following requirements affecting the use of lookups in Progress Dynamics Web applications:

- **Availability of target entities** — Entities that are the object of a dynamic lookup must be imported into the Repository. Because lookups in Progress Dynamics Web applications use SDOs for displaying data, and SDOs rely on entities being present in the Repository, a lookup will fail if the target entity has not been imported.
- **Validation of lookup fields** — To ensure that only valid values are accepted and stored when selected from a lookup list in a Web browser, you must add the appropriate business logic to the SDO. The lookup properties `PopupOnAmbiguous`, `PopupOnNotAvail`, `PopupOnUniqueAmbiguous`, and `BlankOnNotAvail` are not implemented for Progress Dynamics Web applications.
- **Display of user selection** — For a field linked to a dynamic lookup to work properly, the field specified as the displayed field for the lookup must also be assigned a browse sequence number, so that it will be retrieved. (For example, in the Smart DataField

Maintenance dialog shown in [Figure 4-4](#), the specified **Displayed name** field, customerName, is assigned browse sequence number 2.) If this is not the case, the Web browser does not populate the field with the user-selected value until the record is saved.

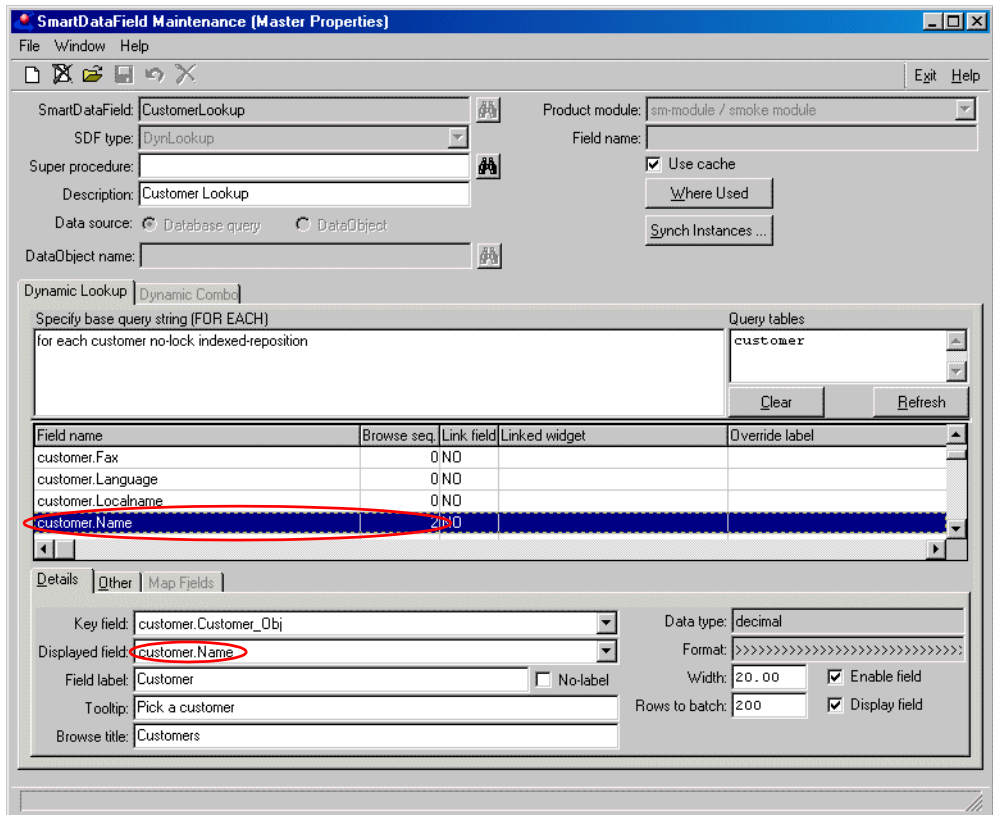


Figure 4-4: Including displayed field in dynamic lookup retrieval

SmartBusinessObjects

A container with visual objects that use an SBO as a data source will function in a Web environment only if a single SDO is specified as the value for each of the following Container Builder fields: **Data Sources**, **Update Targets**, and **Nav target**.

If these values are not set, or if lists of values are specified for them, the container will not work in a Web application.

Popups in SmartViewerObjects

The default value of the ShowPopup property for a SmartViewer object is Yes. Overriding this value for a viewer instance in a container has no effect in a Progress Dynamics Web application. To suppress the popup button for certain data fields, you must change the default at the field level.

Comments

The following Comments features are not supported in Progress Dynamics Web applications:

- AutoComments property
- Comments on comments

TreeView

When developing TreeViews, you should be aware of several considerations for Web applications. This section describes these considerations with reference to the Dynamic TreeView Builder tool, the Tree Node Control tool, and TreeViews whose structure is generated by an extract program.

The following considerations apply to the Dynamic TreeView Builder tool:

- **Layout** — Only the TreeView layout option is supported for Web applications.
- **Filter viewer** — Only dynamic viewers (object type **DynView**) are supported for Web applications.
- **Tree style** — TreeViews in Web applications always appear with tree lines, regardless of the setting chosen for the **Tree style** field.
- **Auto sort** — If this option is enabled, different node types are processed separately and appear in the following order: text nodes, menu nodes (not sorted), extract and data nodes (mixed). For menu nodes, no sorting is applied even if **Auto sort** is checked; the logical menu hierarchy is preserved.
- **Checkboxes** — The use of checkboxes is not supported. Checkboxes defined for a Windows application do not appear in the Web application.
- **Root lines** — Vertical lines always connect multiple root nodes in Web applications, whether the **Show root lines** checkbox is selected or not.

The following considerations apply to the Tree Node Control tool:

- **Checkboxes** — Since checkboxes are not supported for Web applications, the **Node checked** option (which determines the initial appearance of a checkbox) is not meaningful, and its setting is ignored.
- **Parameters for launched data source** — Progress Dynamics Web applications do not support the passing of parameters to the data source that runs when the node is selected. Any value specified in the **Run attribute** field is ignored.
- **Node images** — If you choose to use custom images rather than the installed defaults, for best results use GIF images with a height of 15 pixels.

When you use an extract program to generate TreeView structure, you must populate the `record_ref` field with a specific value from the `gsc_entity_mnemonic` table. The value used depends on whether the application runs as a Web application or a Windows application, as follows:

- **Web application** — The assigned value of `record_ref` must be the value of `entity_key_field`.
- **Windows application** — The assigned value of `record_ref` must be the value of `entity_object_field`.

4.4.3 Unsupported features and objects

This section lists restrictions on development and use of Web applications.

Restrictions for developers

To develop and manage your Web applications, you must use the Progress Dynamics GUI tools on Windows. You cannot use a Web browser interface for the following functions:

- Development
- Administration

Certain object types that are available for use in Windows applications are not supported in Web applications. [Table 4–1](#) lists these objects.

Table 4–1: Unsupported objects

Object type	Notes
Static visual object	You can convert static SmartObjects to dynamic objects by opening them in the AppBuilder and selecting File→Save As Dynamic Object .
SmartSelect object	You can use Dynamic Lookup or Dynamic Combo objects as alternatives to SmartSelect objects.
DynLookup or DynCombo based on temp-table	Lookups and Combos that are based on temp-tables do not work in Progress Dynamics Web applications.
One-to-one SmartBusinessObject	Progress Dynamics Web applications do not support SBOs that are based on schema tables with one-to-one relationships.

Auditing

The auditing function is not available to Web application users. The **Audit** button appears on the browse toolbar but is non-functional.

4.4.4 Browser behavior


Refreshing container data

Users should be advised to manually refresh the application display frequently to be sure of seeing current data. Data is not refreshed automatically on the client, and within a single browser session, even closing and re-launching the application may not refresh the data.

Resizing browse columns

Browse columns cannot be manually resized within a Web browser. A Browse object is implemented as an HTML table. The Web browser automatically resizes browse columns to fit the data.

Filtering lookup field choices

To constrain the initial list of choices for a lookup field, Web users enter the desired starting string (for example, **sa** to show only records whose values start with "sa") and then display the filtered browse by clicking the binocular icon (). In contrast, Windows users enter the starting string and press the **TAB** key. Pressing **TAB** in a Web application advances focus to the next field.

Tree View display and behavior

Progress Dynamics Web applications differ slightly from Windows applications with respect to the display and behavior of TreeViews. [Table 4–2](#) lists the differences.

Table 4–2: TreeView characteristics in Web applications

UI element or user action	Web	Windows
Window title	If a text node is selected, the Web browser window title is the title assigned to the TreeView. If an SDO node is selected, the Web browser window title is the title assigned to the selected container.	In all cases, the application window title is the title assigned to the TreeView.
Menu structure TreeView node images	The Web browser displays the image specified for the menu item in the Toolbar and Menu Designer.	The application displays the image specified for the tree node in the Tree Node Control.
Add first record to empty table	The user must right-click in white space and use the context menu to add the first record.	The viewer comes up in Add mode when the Tree View window is launched.
Right-click in white space	The context menu option adds a root node record.	The context menu option adds a child record to the selected root node.
Right-click on node	The context menu for the node appears, but the node is not selected.	The context menu for the node appears, and the node is selected.

Adding comments

When a user adds the first comment to a record that previously had none, the screen does not display the red comment indicator until the screen is refreshed or the next batch of data is fetched.

4.5 Deployment

Deploying Progress Dynamics Web applications is very similar to deploying Windows GUI applications. For general information about deploying Progress Dynamics applications, see the *[Progress Dynamics Administration Guide](#)*.

The additional considerations regarding the deployment of Progress Dynamics Web applications are:

- The deployment environment must be configured with a Web Server and with WebSpeed Version 3.1D (Progress Version 9.1D). See [Chapter 3, “Setting Up Progress Dynamics for Running Web Applications,”](#) for more information.
- Your Web Server and WebSpeed configuration must have access to the CSS, JavaScript, and image static files. See [Chapter 3, “Setting Up Progress Dynamics for Running Web Applications,”](#) for more information.

Customizing with Dynamic HTML

This chapter explains how Progress Dynamics uses Dynamic HTML (DHTML) to implement a user interface for applications running in a Web browser environment. It also explains how to use DHTML to change the look and feel of the application.

The DHTML technologies described in this chapter are:

- [HTML](#)
- [Cascading Style Sheets](#)
- [JavaScript objects](#)
- [Images](#)

5.1 HTML

This section describes the default HTML files included in Progress Dynamics and how they can be used to customize Progress Dynamics Web applications.

Topics include:

- [Default HTML file](#)
- [Layouts](#)
- [CSS links](#)
- [JavaScript links](#)
- [Page title](#)
- [XHTML conformance](#)
- [Unicode UTF-8 support](#)
- [Application help](#)

5.1.1 Default HTML file

Progress Dynamics includes a default HTML page layout file as well as default HTML help files.

The HTML page layout file (`default.htm`) is located in `install_dir\tty\icf\ry\dhtml`. Typically, you copy and rename `default.htm` and use the copy as a startup page for your Web application. A user accesses your Progress Dynamics Web application by specifying the file in a URL. See the “[URL syntax for Progress Dynamics Web applications](#)” section in [Chapter 4](#), “[Progress Dynamics Web Applications](#),” for more information.

The help files are located in `install_dir\src\icf\ry\dhtml`. There are two: `ryhelp.html` and `ryabout.html`. Both contain generic system information that you can change to create help that is suitable for your application. You must compile the HTML help files into r-code and copy the r-code files to `install_dir\tty\icf\ry\dhtml` in order to deploy them. See the “[Application help](#)” section for more information.

CAUTION: Upgrades to later releases of Progress Dynamics overwrite the files in `install_dir\tty\icf\ry\dhtml`. Any changes to the default page layout or default help files will be lost during installation. If you make any changes to any of these files, you must save your changes before the upgrade, and reapply your changes after the upgrade.

5.1.2 Layouts

As shown in [Figure 5–1](#), the `default.htm` file provides a layout for your Progress Dynamics Web application.

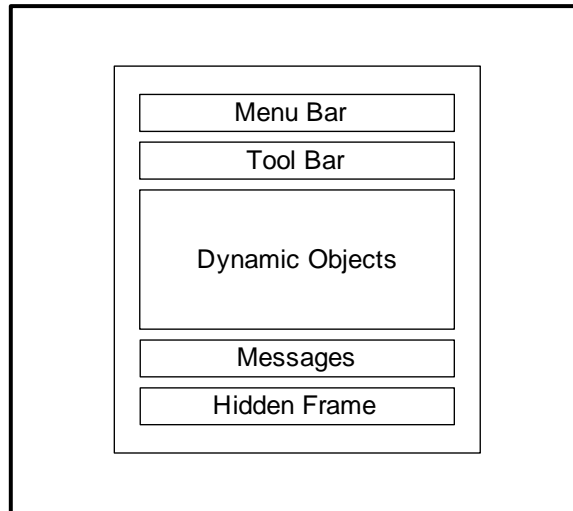


Figure 5–1: The page layout in `default.htm`

The default HTML page provides the layout for the following user interface components:

- **Menu Bar** — Contains general application menus or menus for particular objects within an application.
- **Tool Bar** — Contains general application tools or tools for particular objects within an application.
- **Dynamic Object Area** — Provides an area for displaying the application user interface.
- **Message Area** — Provides an area where client and server messages appear. You can modify the behavior of the message area (for example, you can suppress the display of messages to avoid redundancy caused by automatic popup windows that the browser implements). See the [“Controlling message display”](#) section for instructions.
- **Hidden Frame** — Provides an area to contain client/server communications.

You can change the layout of your Web page by editing the markup of one of the *default.htm* files. However, you should have experience working with HTML and DHTML. You need to understand the consequences of moving, changing, or deleting elements in the default HTML files.

Make your changes in copies of the default files. Do not edit the originals.

Controlling message display

The behavior of the message area is controlled by the following entry in *default.htm* (shown as installed):

```
<div id="info" mode="auto" style="height:100px;width:99%;"></div>
```

There are three valid values for the *mode* attribute:

- **auto** — The message area is not initially visible; it appears and displays the appropriate messages when an error occurs. This is the default (as-installed) setting.
- **on** — The message area and any messages are always visible.
- **off** — The message area appears only when the user clicks the **Toggle Status Window** button, even if errors occur.

Some browsers automatically pop up an alert window when errors occur. If this is the case for your users, you might want to set the *mode* attribute in *default.htm* to **off** to avoid redundancy.

5.1.3 CSS links

The *default.htm* file contains a link to a style sheet for the static Web page (*rymain.css*), as well as a link to a style sheet for the dynamic object area (*ryapp.css*). The following shows the HTML markup for these links:

```
<link rel="stylesheet" type="text/css" href=" ../dhtml/rymain.css" />  
<td id="app" dyn="" css=" ../dhtml/ryapp.css" encoding="UTF-8">
```

Edit these links to create references to your own style sheet files.

For more information, see the “[Cascading Style Sheets](#)” section.

5.1.4 JavaScript links

The default `html` files contain `SCRIPT` elements that call the JavaScript files that implement user interface objects. For example:

```

        .
        .
<script language="Javascript" src="../dhtml/ryhotkey.js" ></script>
<script language="Javascript" src="../dhtml/rymenubar.js" ></script>
<script language="Javascript" src="../dhtml/rytoolbar.js" ></script>
<script language="Javascript" src="../dhtml/rytreeview.js" ></script>
        .
        .

```

You can edit these links to add references to your own JavaScript files.

5.1.5 Page title

The name of the `TITLE` element in the `HEAD` section of the default HTML files is Progress Dynamics, as shown in the following markup:

```
<title>Progress Dynamics</title>
```

Since the content of the `TITLE` element shows up in search engines and appears in the Web browser title bar, you should change it to something that is appropriate to your application.

5.1.6 XHTML conformance

The default HTML layout file and the output from the UI Manager conform to the W3C XHTML standard.

At the beginning of the layout file, there is a Document Type Declaration (DTD) that references the standard:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xml:lang="en" lang="en"
  xmlns="http://www.w3.org/1999/xhtml">

```

To maintain XHTML conformance when you modify copies of the layout file, you should be familiar with the XHTML standard. For example, you should be aware that:

- XHTML, unlike HTML, is case sensitive. For example, `` and `` are not equivalent in XHTML.
- All XHTML tags require matching end tags. For example, `` requires a matching `` in XHTML. In HTML, the `` end tag is optional.

5.1.7 Unicode UTF-8 support

To allow language translation of labels for UI objects like buttons, menu items, and tabs, the default HTML page supports Unicode UTF-8 encoding. UTF-8 is enabled in the `.htm` files within the opening XML declaration:

```
<?xml version="1.0" encoding="UTF-8"?>
```

To enable UTF-8 encoding, you must also:

- Specify the encoding attribute as UTF-8 on the application tag in the `default.htm` file. For example:

```
<td id="app" dyn="" css="../dhtml/ryapp.css" encoding="UTF-8">
```

This markup ensures that all returned pages have `encoding="UTF-8"` set in their XML declarations.

- Set the Progress Dynamics WebSpeed Agent's `-cpinternal` startup parameter appropriately. For more information on agent configuration and startup, see the *WebSpeed Installation and Configuration Guide*.

See the *Progress Internationalization Guide* for more information about Unicode and UTF-8 encoding.

5.1.8 Application help

By default, the **Help** menu in Web applications contains two selections: **Help Topics** and **Help About**. **Help Topics** invokes `..\dhtml\ryhelp.r`. **Help About** invokes `..\dhtml\ryabout.r`.

Figure 5–2 shows the default help dialog box displayed when you select either **Help Topics** or **Help About** from your application. As you can see, the content is general system information.

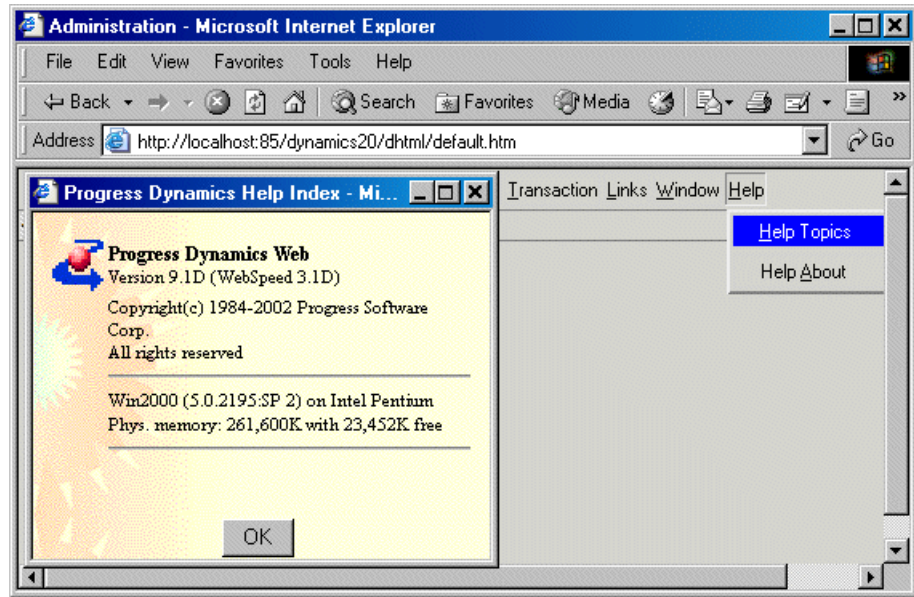


Figure 5–2: Default help topic

To create help that is more appropriate to your Web application, follow these steps:

- 1 ♦ Copy `ryhelp.html` and `ryabout.html` from `install_dir\src\icf\ry\dhtml\` to `install_dir\tty\icf\ry\dhtml\ryhelp.html`.

These files are templates for your application help. You can also start with an empty file and create your help from scratch.

- 2 ♦ Edit `ryhelp.html` and `ryabout.html` in a Text editor or an HTML editor and save.

If you started with an empty file rather than the templates, you must use the filenames `ryhelp.html` and `ryabout.html` when you save. Also, be sure you save your files in `install_dir\tty\icf\ry\dhtml\`.

- 3 ♦ Open WebSpeed WebTools.

WebSpeed WebTools are a browser-based set of application development and management tools. For more information, see the *WebSpeed Developer's Guide*.

- 4 ♦ Select **File Tools** from the **Tools** menu.

- 5 ♦ Navigate to the directory that contains your `ryhelp.html` and `ryabout.html` source files.

- 6 ♦ Select and compile your help files.

A compile icon is located on a toolbar at the top of the **File Tools** page.

Compilation creates the r-code files, `ryhelp.r` and `ryabout.r`, which will be invoked from the **Help** menu.

5.2 Cascading Style Sheets

This section gives an overview of Cascading Style Sheets (CSS) and explains how they can be used to customize Progress Dynamics Web applications. Topics include:

- [The default CSS files](#)
- [Overview of CSS rules](#)
- [Customizing with CSS](#)
- [Applying your own style sheets](#)
- [Applying themes for groups of users](#)

NOTE: Working with CSS to apply customizations to your Progress Dynamics Web applications requires more information than you will find here. If you are not familiar with CSS, a good resource for learning about it is the World Wide Web Consortium site at <http://www.w3.org/Style/CSS/learning>. It contains lists of books, online sources, and user groups.

5.2.1 The default CSS files

The two CSS files that are included with the Progress Dynamics Web framework are:

- **rymain.css** — Used for static Web pages. It defines styles for toolbars, tab folders, menus bars, and tree view menus.
- **ryapp.css** — Used for Dynamics application objects. It is included by the UI Manager in all output sent to the Web browser. This file defines styles for WebDataObjects (client-side SDO proxies), toolbars, tab folders, browsers, viewers, and fields.

The default HTML layout page (default.htm) links to `rymain.css` and `ryapp.css`. See the [“Applying your own style sheets”](#) section for information about these links.

5.2.2 Overview of CSS rules

CSS files contain a list of rules that apply formatting to elements in an HTML page. Each rule has a selector and a declaration. The declaration contains property and value pairs. [Figure 5–3](#) shows a simple rule that establishes the font size for all paragraph elements (<P>) in an HTML page.

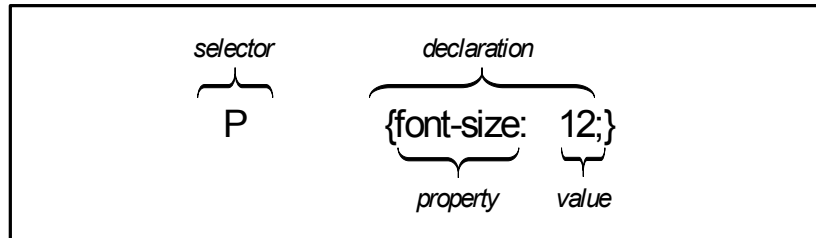


Figure 5–3: CSS rule example

CSS supports three basic types of selectors. You can see the three basic types in `rymain.css`:

1. **HTML Element Selector** — Defines a style for all elements of a particular type. The selector is usually an HTML tag without the brackets (< >).

For example, the `BODY` HTML element selector in `rymain.css` defines the background color and font attributes:

```
BODY    {background-color:#D6D3CE;font-color:#000000;
         font-family:MS Sans Serif;font-size:8pt;margin:0;}
```

2. **Class Selector** — Defines a style set in a `CLASS` attribute. It is always preceded by a period (.). Class selectors are called by setting the `CLASS` attribute in an element tag. They can assign styles to any of the elements in an HTML page (unlike the HTML Element Selector, which assigns styles to one particular element type).

For example, `rymain.css` defines the following `.hide` selector, which renders elements invisible on the HTML page:

```
.hide    {display:none;}
```

The `.hide` selector appears in `iframe`, `div`, and `form` elements, as shown in the following fragment from `default.htm`:

```

        .
        .
<iframe name="win0" border=0 frameborder="no" class="hide"
application="yes" onload="if(window.fLoad) fLoad()"></iframe>
        .
        .
        .
<div id="info" class="hide" style="height:100px;width:99%;"></div>
        .
        .
        .
<form name="form" method="post" target="run" action="" class="hide">
        .
        .
```

3. **ID Selector** — Defines a style set in an ID attribute. It is always preceded by a pound sign (#). Unlike an HTML element or a class selector, an ID selector should be used only once on an HTML page. Also, the style defined in an ID selector overrides any other style setting.

In Progress Dynamics, ID selectors usually apply styles to UI objects that are grouped under span or div elements.

For example, `rymain.css` defines the following `#toolbar` selector, which defines the style of a toolbar object:

```
#toolbar    {margin:0px;border-bottom:thin groove;
              border-top:thin groove;width:99%;height:28px;}
```

In `default.htm`, the formatting is applied in the following div element:

```
<div id="toolbar" imgdir="../img"></div>
```

In addition to rules using basic selector types, `rymain.css` also contains rules that use selectors that match elements based on parent-child relationships. The selector in this type of rule is called a **Contextual Selector**.

For example, the selector in the following rule in `rymain.css` matches TD (table cell) elements that are descendants of an element whose ID attribute is `toolbar`:

```
#toolbar TD  {margin:0;padding:1px 3px 1px 3px;border:1px #D6D3CE solid}
```

The selector in the following rule, also in `rymain.css`, shows the use of the greater-than symbol (>) to limit matching to direct descendants:

```
#menubar > TABLE  {margin:0pt;padding:0pt;background:#D6D3CE;}
```

It is beyond the scope of this manual to go into more detail on CSS rules. If you implement style changes in your Progress Dynamics Web applications, you need to be more familiar with the CSS specification. In particular, you need to understand how CSS establishes precedence when a number of rules apply to the same element.

Again, if you need more information about CSS, a good source is the World Wide Web Consortium site at <http://www.w3.org/Style/CSS/learning>. It contains lists of books, online sources, and user groups.

5.2.3 Customizing with CSS

You can use `rymain.css` or `ryapp.css` as templates for your own style sheets. If you want to use them as templates copy, rename, and edit the copy.

Do not edit `rymain.css` or `ryapp.css` directly. Upgrades to later releases of Progress Dynamics will overwrite `rymain.css` and `ryapp.css`. So, any changes will be lost after the upgrade.

There are examples of customized versions of `rymain.css` and `ryapp.css` in `install_dir\tty\icf\ry\dhtml`. Look at `snow_main.css` and `snow_app.css` to see how style sheets can be changed to modify the overall look of your application.

After you have created your own style sheets, see the [“Applying your own style sheets”](#) section for information on how to implement them.

5.2.4 Applying your own style sheets

There are two methods for applying your own CSS files to your Progress Dynamics Web application:

1. Edit the `default.htm` layout file and change the style sheet links so that they point to your own CSS files.

The following example shows the HTML markup for these links:

```
<link rel="stylesheet" type="text/css" href="..dhtml/rymain.css" />  
<td id="app" dyn="" css="..dhtml/ryapp.css">
```

Change the markup displayed in bold so that the links point to your own CSS files.

2. Add the name of your CSS file to the `StyleSheetFile` attribute in a property sheet.

Use this method when you want to override the style sheets linked to in the `default.htm` layout file to apply a style to a particular object.

The files specified in the `StyleSheetFile` attribute override the style sheet files specified in the `default.htm` layout file.

You can specify a number of files in the `StyleSheetFile` attribute. The order in which you add CSS files is important. The last file in the list overrides the settings in preceding files.

To associate your own CSS file with your Web application, add the filename to the `StyleSheetFile` attribute. The `StyleSheetFile` attribute is listed in the Dynamics Property Sheet dialog box. If you want to apply a CSS file to a particular object, open the object in AppBuilder and then select **Window → Dynamic Property Sheet**. If you want to apply a CSS file to a container, open the container in the Container Builder and then select **Advanced → Properties**.

As shown in [Figure 5–4](#), the value for `StyleSheetFile` is a comma-delimited list of CSS files. The UI Manager adds a reference to each CSS file in the list to the HTML header of the Web application page that it sends to a client.

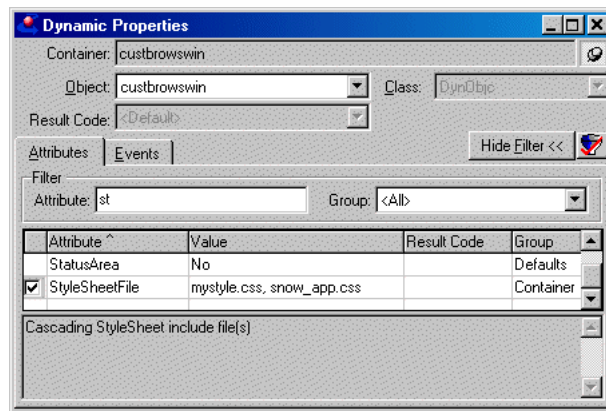


Figure 5–4: `StyleSheetFile` in the Dynamic Properties dialog box

5.2.5 Applying themes for groups of users

Themes are the sum of all the graphical elements and styles that give an application a distinctive look. You can implement themes for specific user groups by creating multiple instances of the default HTML layout file and linking to a different style sheet in each instance.

For example, you could implement a different theme depending on whether the user is a customer or a sales person. First, make two copies of `default.htm`, calling one `cust.htm` and other `sales.htm`. Then edit the style sheet links to call the CSS files tailored for each group. The markup in `cust.htm` might look something like the following:

```

      .
      .
<link rel="stylesheet" type="text/css" href=" ../dhtml/cust_main.css" />
      .
      .
<td id="app" dyn="" css=" ../dhtml/cust_app.css">
```

Users from each group access the application by specifying the appropriate URL. For example, the customer URL for an intranet application might look something like the following:

```
http://myhost/dynamics21/cust.htm?icfobj=oeCustBrowseWin
```

5.3 JavaScript objects

This section gives an overview of JavaScript objects and explains how they can be used to customize Progress Dynamics Web applications.

Topics include:

- [Summary of JavaScript object files](#)
- [Customizing with JavaScript](#)
- [Extending existing JavaScript object types](#)
- [Adding your own functions](#)

5.3.1 Summary of JavaScript object files

JavaScript objects contain JavaScript functions that implement application object behaviors in Progress Dynamics Web applications.

Unlike simple JavaScript, JavaScript objects expose methods, events, and parameters to other objects. They support the client-side behaviors, interface objects, events, and validation that allow Web pages to emulate standard Windows/GUI functionality.

NOTE: The JavaScript objects in Progress Dynamics conform to the W3C DOM standard.

The default HTML layout file links the JavaScript object files to your Progress Dynamics Web application. (See the [“JavaScript links”](#) section for more information.) Like the other static files, they are installed in `install_dir\tty\icf\ry\dhtml`.

[Table 5–1](#) lists the JavaScript object files and indicates their purpose.

Table 5–1: JavaScript object files

(1 of 2)

Object type	Filename	Objects implemented
Generic	ryapph.js	The window manager.
	ryinfo.js	The Info object (message area).
	ryinithtml.js	Initializations for the <code>default.htm</code> files.
	rylookup.js	Dynamic lookup objects.
	rylogic.js	Implements the client logic API. (See the “The Dynamics Images utility” section for more information.)
	ryrender.js	A collection of screen rendering utilities that handle object positioning within an application screen.
	ryutil.js	A collection of generic utilities.

Table 5–1: JavaScript object files (2 of 2)

Object type	Filename	Objects implemented
Menu	ryhotkey.js	Support for hot keys (shortcut keys).
	rymenubar.js	Menu bar.
	rypopup.js	Pop-up menus.
	rytoolbar.js	Toolbar.
	rytreeview.js	TreeView.
ADM2 Simulation	rybrowse.js	Browser (similar to the ADM2 SmartDataBrowse).
	rydata.js	Implements a data object that provides data caching on the client.
	ryfolder.js	Tabbed folder (similar to the ADM2 SmartFolder).
	rywbo.js	WebBusinessObject (manages visual and data objects on the client and also posts data to the server).
	rywdo.js	WebDataObject (client-side data manager).

5.3.2 Customizing with JavaScript

You can customize the Progress Dynamics DHTML framework by extending existing objects (see the [“Extending existing JavaScript object types”](#) section) and by creating your own client-side actions (see the [“Adding your own functions”](#) section).

You can use some of the default .js files in *install_dir\qty\icf\ry\dhtml* as templates for your customizations. However, do not edit any of the default .js files. Upgrades to later releases of Progress Dynamics will overwrite these files. So, any changes you make will be lost after the upgrade. It is safer to copy the default file, rename it, and edit the copy.

5.3.3 Extending existing JavaScript object types

You can customize the Progress Dynamics DHTML framework by extending objects with your own JavaScript code. You can do this by configuring existing objects to use alternative code for object instantiation. To do this, you must know JavaScript fairly well and must be able to read the current code.

You might also have future compatibility issues if the APIs change. Generally, extending existing objects is safer than replacing them. Also, since ADM objects expect a certain API, the best method for extending JavaScript object types is to make modifications based on an existing object type.

For example, consider the Browse object that is defined in `rybrowse.js`. To extend the Browse object, follow these steps:

- 1 ♦ Create a new file (for example, `mybrowse.js`) in the `install_dir/tty/icf/dhtml/`.

The default `rybrowse.js` is also found in `install_dir/tty/icf/dhtml/`.

- 2 ♦ Include your file by adding a `SCRIPT` element to your default HTML file. For example:

```
<script language="javascript" src="mybrowse.js"></script>
```

- 3 ♦ Define an object prototype that is similar to the object defined in `rybrowse.js`.

The object prototype name must have the first letter uppercase and the rest in lowercase. See `Browse2` in the following example:

```
// mybrowse.js
// Sample of syntax for overriding current Browse code with new
functionality

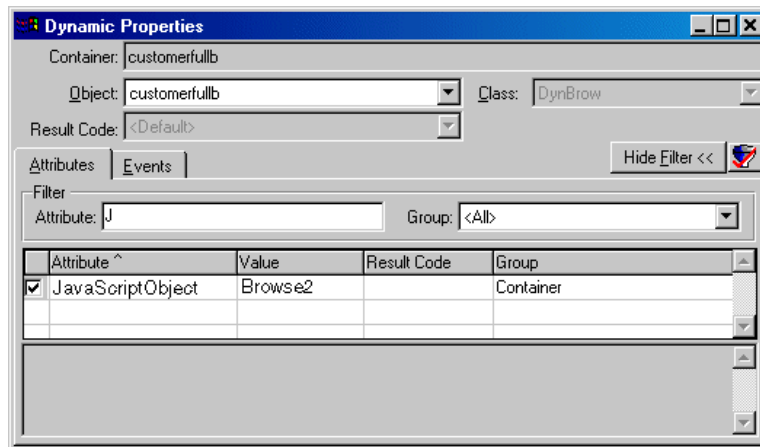
Browse2.prototype=new Browse(); //To override/extend the current Browse
Browse2.prototype.instance_myvar=value; //Add your own variables

function Browse2(){ //Code executes when created, but after Browse()
{initialization code...}
}

Browse2.prototype.myfunc=function(){ // Add your own methods
{my code...}
}
```

- 4 ♦ Specify your object prototype name as a `JavaScriptObject` attribute for the application object.

The `JavaScriptObject` attribute is one of the attributes that you can set in the Dynamic Property dialog box. For example, the following shows a setting that causes `Browse` to be overridden by `Browse2` in the `customerfullb` application object:



When the application handler object initiates `customerfullb`, it will see the `Browse2` object type and initiate it with the `Browse2` prototype.

5.3.4 Adding your own functions

Progress Dynamics includes a JavaScript API, which contains a variety of client-side actions that are useful for creating your own functions.

For example, the following code fragment is a function that uses the JavaScript API to refresh data from the DynSports customer table:

```
// Sample Javascript include for Client-side Custom logic

// refresh customer data every 10 seconds
window.setInterval('customerfullo_refresh()',10000);

function customerfullo_refresh(){
    if(window.main.app==window &&
    apph.action('customerfullo.handle').updatemode=='view'){
        // Only run if currently in display and in view mode
        apph.actions(['server.customerfullo.refresh','wbo.submit']);
    }
}
```

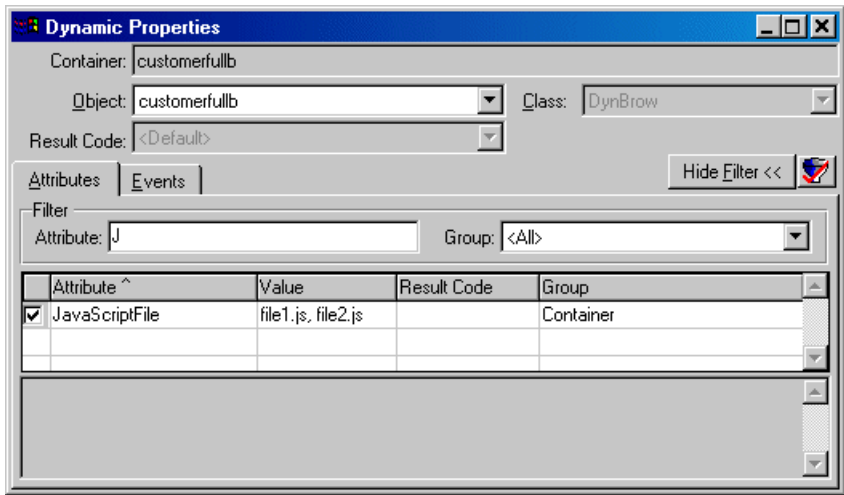
See [Appendix A, “JavaScript API Reference,”](#) for descriptions of individual actions, syntax information, and examples.

To implement your JavaScript functions:

- 1 ♦ Save your JavaScript code to a file in the *install_dir\tty\icf\ry\dhtml* directory or to a directory that your Progress Dynamics Broker can find.

- 2 ♦ Add your filename to the JavaScriptFile attribute in the Dynamics Property dialog on the class, master, or instance level of your object. (Normally, you would avoid overrides on the base or class level since all objects in the class would inherit your changes.)

The following figure shows that the values for the JavaScriptFile attribute is a comma-separated list of filenames:



5.4 Images

Progress Dynamics includes a number of images of common user interface graphics in *dynamics_install_dir\tty\icf\ry\img*. You can add your own images to this directory or modify the default images. However, be aware that if you modify any of the images you will lose your changes when you upgrade to a later version of Progress Dynamics.

As shown in [Figure 5–5](#), you can view the default images with the Dynamics Images utility. You can access the Dynamics Images utility from the Web Test Page. The page is the default.htm file that is in *dynamics_install_dir\tty\icf\ry*.

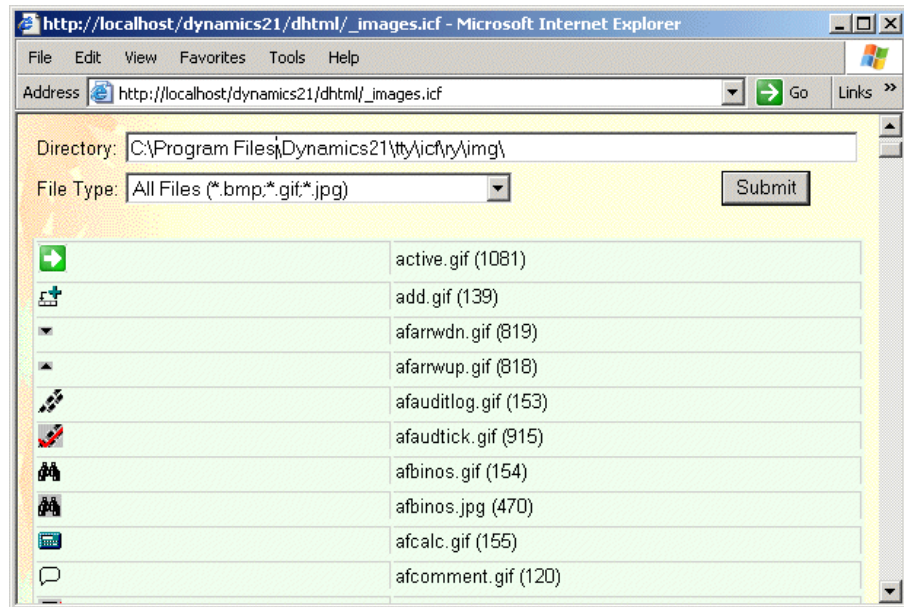


Figure 5–5: The Dynamics Images utility

Server-side Business Logic and Client Logic

This chapter describes how to use server-side business logic and client logic in Progress Dynamics Web applications. You can use the information in this chapter to customize your Web application. Also, you should review the material in this section if you need to port existing 4GL UI logic to the Progress Dynamics web application framework.

The major sections in this chapter are:

- [Server-side business logic](#)
- [Client logic](#)

6.1 Server-side business logic

The recommended deployment of business logic is to execute it on the server side. The primary advantage is that you avoid the necessity of writing multiple platform-specific, client-side procedures. You implement business logic as an SDO data logic procedure, as a Progress Dynamics PLIP (Persistent Library of Internal Procedures), or as a structured procedure.

You use JavaScript and the JavaScript API (see [Appendix A, “JavaScript API Reference”](#)) on the DHTML client to retrieve data from the business logic on the server. For example, the following code example retrieves the current value of the Date Format and Date Format Mask fields and displays them in a JavaScript Alert box:

```
function dateAlert() {  
  var cVal1 = apph.action('gscgcfullo.date_format.get');  
  var cVal2 = apph.action('gscgcfullo.date_format_mask.get');  
  alert("Format = " + cVal1 + "\n" + "Mask = " + cVal2);  
}
```

6.1.1 Invoking server-side business logic from the client

If you want server-side business logic to respond to some event on the DHTML client, you must define JavaScript event handlers for dynamic viewer fields. After creating the event handler and saving it to a file, you must add its filename to the JavaScriptFile attribute for the container. (See the “Customizing with JavaScript” section in [Chapter 5, “Customizing with Dynamic HTML.”](#))

The event can be any supported event for the viewer field type. For example:

```
function blctest(val1, val2) {  
  var call, target, flags, params;  
  call = 'proc1'; // procedure or function to run  
  target = 'ry\test\foo.p'; //program to run  
  flags = 's'; // Dynamic Call Wrapper flags  
  params = "input character '" + val1 + "'\t" + "input integer '" + val2 + "'";  
  
  apph.runOnServer(call,target,flags,params);  
}
```

The `apph.runOnServer` function is a client-side equivalent to a server-side interface to the Dynamic Call Wrapper. For more information about the Dynamic Call Wrapper, see the [Progress Dynamics ADM2 API Reference](#).

I/O blocking

Progress Dynamics Web applications do not support user prompts or messages within business logic. Therefore, I/O blocking statements (where execution is suspended until some user action is completed) should be avoided in server-side business logic.

Output parameters

Business logic containing output parameters is not supported for Progress Dynamics Web applications. If your business logic contains output parameters, they must be wrapped within additional 4GL that can deal with output parameters.

6.1.2 Invoking JavaScript APIs from the server

You use the UI Manager's `setClientAction` procedure to invoke JavaScript APIs from the server. Typically, you use `setClientAction` to implement a UI change on the DHTML client in response to some condition created by the server-side business logic.

For example, the following code hides the `address2` field in the `customerfullo` WDO.

```
gshUIManager = DYNAMIC-FUNCTION("getManagerHandle":U IN THIS-PROCEDURE,  
"UserInterfaceManager":U).  
RUN setClientAction IN gshUIManager ('customerfullo.address2.hide').
```

The `setClientAction` procedure runs separately for each WDO. (See [“WebDataObject \(wdo\)”](#) in [Appendix A, “JavaScript API Reference”](#) for a list of WDOs.) Calls to `setClientAction` are queued until the end of the web request, at which time they are sent to the DHTML client as part of the response data stream.

6.1.3 Context management

Since the Web server does not maintain context between Web requests, you must implement storing and retrieving context information in your 4GL code.

To store context information on the server between Web requests, use the `setPropertyList` function in the Session Manager API. This function has the following input parameters:

- **pcPropertyList** — A comma-delimited list of the names of the properties to be set.
- **pcPropertyValues** — A CHR(3)-delimited list of corresponding property values.
- **p1SessionOnly** — A logical. If set to YES, maintain property values only for the duration of the session.

The following shows the usage:

```
DYNAMIC-FUNCTION("setPropertyList":U IN gshSessionManager, pcSDOName,  
pcSDOInfo, NO).
```

To retrieve context information on the server between Web requests, use the `getPropertyList` function in the Session Manager API. The function returns a CHR(3)-delimited list of property values. This this function has the following input parameters:

- **pcPropertyList** — A comma-delimited list of the names of properties to be retrieved.
- **pISessionOnly** — A logical. If set to YES, maintain property values only for the duration of the session.

The following example shows the usage:

```
DYNAMIC-FUNCTION("getPropertyList":U IN gshSessionManager, pcSDOName, NO).
```

To store temporary context information on the server only for the duration of the Web request, use the `setProperty` function in the Web Request Manager API.

To add to a list of temporary context information on the server only for the duration of the Web request, use the `addProperty` function in the Web Request Manager API. This function is useful for accumulating lists of UI changes that are sent to the client at the end of a Web request.

To retrieve temporary context information on the server that has been stored using the `setProperty` function, use the `getProperty` function in the Web Request Manager API.

For more information on these functions, see the [Progress Dynamics Managers API Reference](#).

6.2 Client logic

Client logic is implemented through the client logic API. The client logic API is a collection of functions that extend the visual class, and a programming standard that supports easier development of client-side user interface logic in containers, browses, viewers, and other visual objects.

The client logic API essentially supports widget manipulation in containers, browses, viewers, and visual objects in both static and dynamic objects. For example, you might want to enable some fields in a dynamic viewer based upon the value of another field. The API, in particular, makes it much easier to associate code with one browse or viewer and have that code manipulate widgets in a different browse or viewer (as long as both are in the same container).

You can use the client logic API in static object code or in custom super procedures for dynamic objects.

Note that an application for both the GUI and DHTML clients must maintain two versions of client logic, one written in 4GL for GUI rendering and another written in JavaScript for Web browser rendering. This chapter describes the client logic API for DHTML applications. For information on the 4GL applications, see the [Progress Dynamics Programming Handbook](#).

A JavaScript file named `install_dir\tty\icf\ry\dhtml\rylogic.js` implements the API. The default HTML file contains a reference to `rylogic.js`. The Logic object is the internal definition of this file. As with other JavaScript objects supplied with Progress Dynamics, you can extend the Logic object in your own JavaScript file.

6.2.1 Logic object

The Logic object implements the client logic API. Because of this, prefix all references to the APIs with **logic.** to correctly run the proper functions in the Logic object. For example:

```
lRet=logic.enableWidget('customerviewv.name');
```

At design time, you can attach a JavaScript file to a container, browse, or viewer using the JavaScriptFile attribute found in the Dynamics Property Sheet. The DHTML client then associates custom business logic, including client logic, in the attached file with the Web page as a whole and not the object.

NOTE: This is not a limitation of the Dynamics DHTML client implementation, but rather a characteristic of JavaScript and its relationship to the page that includes it.

6.2.2 Object qualification

Qualification in the DHTML client is quite different from the GUI client. Because of the way JavaScript works, all logic is associated with the page—not individual objects. Therefore, a client logic API call does not have a run-time association with the object that the code is associated with during design time. (Thus, the “self” qualifier would not be useful and is not supported in the DHTML client.)

Unqualified names

You should only use qualified names with DHTML client code. One reason for this recommendation is that the DHTML client cannot discriminate between different occurrences of the same data field. Therefore, unqualified names affect every occurrence of the data field on the page.

NOTE: The DHTML client does not support unqualified names at the container level.

Qualified instance names

The basic form of qualification in the DHTML client uses the following syntax:

```
objectname.fieldname
```

Here, *objectname* is the browse or viewer instance name and *fieldname* is a field in that object. For example:

```
orderfullb.custnum  
customerviewv.name
```

Qualification with the browse keyword

You can use the **browse** keyword to specify the browse associated with a particular WDO. The syntax is:

```
wdoname.browse.fieldname.
```

Here, *wdoname* is the data source name, **browse** is a keyword, and *fieldname* is a column in the browse object for that WDO. For example:

```
customerfullo.browse.custnum
```


6.2.3 Client logic functions

[Table 6–1](#) contains a list of the client logic functions that can be used for DHTML applications. For complete information, including examples, see the [Progress Dynamics ADM2 API Reference](#). Information about these functions is also available in the Progress Dynamics AppBuilder online help.

Table 6–1: List of client logic API functions

(1 of 2)

Function	Description
assignFocusedWidget	Sets focus to the named object.
assignWidgetValue	Takes the name of one object and a character screen value as input and sets the SCREEN-VALUE of the object.
assignWidgetValueList	Takes the name of one or more objects and character screen values and a delimiter as input and sets the SCREEN-VALUE of the objects.
blankWidget	Blanks the SCREEN-VALUE of the objects in the namelist.
disableWidget	Disables the objects identified in the namelist.
enableWidget	Enables the objects identified in the name list.
formattedWidgetValue	Returns the SCREEN-VALUE of the object, or in the case of a browse column when in a ROW-DISPLAY trigger, the STRING-VALUE from the RowObject buffer field.
formattedWidgetValueList	Takes the name of one or more objects and returns the SCREEN-VALUE of the object or objects.
hideWidget	Hides the objects identified in the namelist (and their pop-up buttons).
highlightWidget	Sets the background and foreground colors (FGCOLOR and BGCOLOR widget attributes) of the named objects to a standard highlight color, depending on the value of the highlightType argument.
toggleWidget	Reverses the value of one or more objects of type LOGICAL in the name list.
viewWidget	Views the objects identified in the namelist (and their popup buttons).

Table 6–1: **List of client logic API functions** (2 of 2)

Function	Description
widgetHandle	Returns the handle of the requested object. CAUTION: Future versions of Progress Dynamics might not automatically migrate calls to this function. Try to avoid using it. If you do use it, keep track of its use so that you can quickly locate potential migration issues later.
widgetIsBlank	Returns TRUE if the widget is blank, otherwise FALSE.
widgetIsTrue	Returns TRUE if the value of a LOGICAL object is TRUE, otherwise FALSE.
widgetValue	For most objects, returns the INPUT-VALUE of the object.
widgetValueList	Takes the name of one or more objects and a delimiter and returns the INPUT-VALUE of the object.

JavaScript API Reference

This appendix describes the JavaScript API, which can be used for customizing Web applications. Contents include:

- [JavaScript actions](#)
- [JavaScript for client-side functions and commands](#)

For information on how to implement your JavaScript code, see the “[JavaScript objects](#)” section in [Chapter 5](#), “Customizing with Dynamic HTML.”

A.1 JavaScript actions

JavaScript Actions in Progress Dynamics are a number of built-in events that you can use when creating JavaScript customizations for your Web applications.

A.1.1 Syntax

JavaScript Actions in Progress Dynamics are called as quoted string arguments to `apph.action()` or `apph.actions()`. The following example shows the general syntax:

```
apph.action("action_name|parameter|parameter| . . .")
apph.actions(["action_name|parameter", "action_name|parameter", . . .])
```

`apph.action()`

Function name that takes a single action as its argument.

`apph.actions()`

Function name that takes multiple actions as its argument.

action_name

One of the actions described in the sections that follow. The action name is composed of a prefix followed by an action type; for example, `app.back`, `info.clear`. Some action names include the ID of an object; for example, `d1g.object_id`, `main.object_id.disable`.

parameter

A value or values that modify the usage of an action. The actions that take parameters are indicated in the sections that follow.

Differences from 4GL conventions

The syntax descriptions for JavaScript Actions differ from standard 4GL conventions. In JavaScript Action syntax:

- The pipe character (`|`) separates parameters.
- Square brackets (`[]`) enclose an array of actions.

In 4GL syntax statements, the pipe character and square brackets are not literals, but are used to indicate choices and optional parameters.

A.1.2 Application (app)

The `app` action applies to application objects that appear in the dynamic object frame of the default HTML layout page (`default.htm`).

The following table describes the `app` action:

Action	Description
<code>app.exit</code>	Exits from the current screen and goes back to the parent dialog box or the initial screen.

A.1.3 Dialog (dlg)

The `dlg` action executes an application object so that the Web browser's back button returns to the parent object in the Web application. Compare with [“Program \(prg\).”](#)

The following table describes the `dlg` action:

Action	Description
<code>dlg.object_name</code>	Executes the application object or HTML filename specified by <i>object_name</i> .

A.1.4 Information (info)

These actions apply to the Messages frame in the default HTML page layout.

The following table lists and describes the info actions:

Action	Description
<code>info.alert short long arg, arg, ...</code>	Adds a message to the status window with an alert.
<code>info.clear</code>	Clears the messages in the info object.
<code>info.confirm short long action arg, arg, ...</code>	Asks for user confirmation. If OK, then runs <i>action</i> .
<code>info.field short long field arg, arg, ...</code>	Marks a field for error and displays a message.
<code>info.msg short long arg, arg, ...</code>	Adds a message to the status window.
<code>info.prompt short long action arg, arg, ...</code>	Asks for user input, then runs <i>action</i> .
<code>info.yesno short long yes-action no-action cancel-action</code>	Creates a non-modal dialog that requires user response. Runs the first action (<i>yes-action</i>) if the user chooses Yes . Runs the second action (<i>no-action</i>) if the user chooses No . Quits if the user chooses Cancel .
<code>main.info.hide</code>	Hides the info object.
<code>main.info.load([short long, short long ...])</code>	Loads indexed messages.
<code>main.info.show</code>	Shows the info object.

Info Action Parameters

short

A character string that specifies a brief version of a message.

Can also be used as an index to look up messages stored in the information object. Use as an index requires:

- The *long* parameter is blank.
- Loading of messages, using `main.info.load`, must be performed after login.

long

A character string that specifies a more detailed version of a message than the *short* character string.

The character string can optionally contain substitution arguments in the form `&n` where *n* is an integer between 1 and 9 inclusive. Substitutions are performed based on the values specified in the *arg* parameter.

field

The field name qualified by an SDO name (*sdo_name.field_name*).

arg, arg, ...

A comma-separated list of constants, field names, variables, or expressions that result in a character string value. These argument values replace substitution parameters in *long*.

NOTE: The pipe character (|) is used to separate parameters. See the following example.

Example

This example creates an error message for the state field while validating a customer record:

```
apph.action("info.field|code invalid|The &1 code needs to be a valid two  
character US State abbreviation|customer.state");
```

A.1.5 Main (main)

These actions apply to the objects defined in `rymain.css`. The objects include the menu bar, toolbar, tree view menu, and the tab menu.

The following table lists and describes the main actions:

Action	Description
<code>main.object_id.hide</code>	Makes the object disappear.
<code>main.object_id.show</code>	Makes the object visible.

A.1.6 Program (prg)

This action launches an application object where reference to the parent object is not maintained. The Web browser's back button returns to the previous Web site visited and not to the prior page in the Web application. Compare with [“Dialog \(dlg\).”](#)

The following table describes the prg action:

Action	Description
<code>prg.program_name</code>	Executes the application object (file name or object name) specified by <i>program_name</i> .

A.1.7 Server (server)

This action specifies a command to send to the server in the next request.

The following table describes the server action:

Action	Description
<code>server.event</code>	Executes the event specified by <i>event</i> .

Examples

If `wbo.submit` is included, the commands are sent immediately. For example:

```
apph.actions(["server.event", "server.event", "wbo.submit"]);
```


Otherwise, the commands are stored and sent when `wbo.submit` is encountered elsewhere in the application. For example:

```
apph.action("server.event");
apph.action("server.event");
. . .
apph.action("wbo.submit");
```

A.1.8 Tool (tool)

These actions affect a tool's availability and apply to all instances of the tool specified by *tool_id*.

The following table lists and describes the `tool` actions:

Action	Description
<code>tool.tool_id.disable</code>	Grays out the tool.
<code>tool.tool_id.enable</code>	Enables the tool.
<code>tool.tool_id.hide</code>	Makes the tool invisible.
<code>tool.tool_id.show</code>	Displays a hidden tool.
<code>tool.tool_id.toggle</code>	Alternates the tool between enable and disable.

Examples

In the following example, save functionality is disabled in `customerfullo`:

```
apph.action('tool.customerfullo.save.disable');
```

In the following example, save functionality is enabled in `customerfullo`:

```
apph.action('tool.customerfullo.save.enable');
```

In the following example, the info message area is disabled if previously enabled, or enabled if previously disabled:

```
apph.action('tool.main.info.toggle');
```

A.1.9 Utilities (util)

This action launches a pop-up utility (a calculator or a calendar, for example) as a modal pop-up dialog box.

The following table describes the `util` action:

Action	Description
<code>util.name</code>	Launches the utility specified by <i>name</i> .

Example

The following example starts the calculator found in your static DHTML directory:

```
apph.action("util.calculator.htm");
```

A.1.10 WebBusinessObject (wbo)

These actions implement general screen commands and communication with the server. The following table lists and describes the wbo actions:

Action	Description
<code>wbo.submit</code>	Submits the data. Used with a server action. (See “ Server (server) .”)
<code>wbo.commit</code>	Assembles data and submits it to the server.
<code>wbo.undo</code>	Undoes changes to all data.
<code>wbo.refresh</code>	Refreshes all currently displayed data.
<code>wbo.init</code>	Refreshes menus and all input fields for the screen.
<code>wbo.page.changepage</code>	Changes current Tab-folder page.
<code>wbo.rowIndex.node_name.dyntree</code>	For an SDO TreeView node, gets data for the specified <i>node_name</i> ¹ and inserts the data at the specified <i>rowIndex</i> ² .
<code>wbo.rowIndex.band_name.mnu.dyntree</code>	For a menu structure TreeView node, gets data for the specified <i>band_name</i> ³ and inserts the data at the specified <i>rowIndex</i> ² .
<code>wbo.rowIndex.node_name.prg.dyntree</code>	For an extract program TreeView, gets data for the specified <i>node_name</i> ¹ and inserts the data at the specified <i>rowIndex</i> ² .

¹ To specify the node, use the node name as defined in the Tree Node Control tool.

² To specify the row, use the row number from the HTML table for the TreeView.

³ To specify the menu band, use the band code as defined in the Menu and Toolbar Designer tool.

A.1.11 WebDataObject (wdo)

These actions implement client-side data processing.

The following table lists and describes the wdo actions:

(1 of 2)

Action	Description
<i>wdo_name.add</i>	Creates a new record with default values.
<i>wdo_name.cancel</i>	Cancels changes to the current row.
<i>wdo_name.commit</i>	Assembles all changes ready to be submitted.
<i>wdo_name.copy</i>	Creates a new record, but copies values from the currently selected record.
<i>wdo_name.delete</i>	Deletes a record and confirms.
<i>wdo_name.deleterec</i>	Deletes a record, but does not confirm.
<i>wdo_name.export</i>	Exports data to Excel.
<i>wdo_name.filter.disable</i>	Hides filter options.
<i>wdo_name.filter.enable</i>	Displays filter options.
<i>wdo_name.find</i>	Assembles search values and submits them.
<i>wdo_name.first</i>	Calls the first batch of data from the client-side data cache.
<i>wdo_name.field.focus</i>	Sets focus to the named input field.
<i>wdo_name.field.get</i>	Gets the data value displayed on the screen.
<i>wdo_name.field.old</i>	Gets the data value prior to the one displayed on the screen.
<i>wdo_name.handle</i>	Gets the JS object handle to the wdo_name.

<i>wdo_name.last</i>	Calls the last batch of data from the client-side data cache.
<i>wdo_name.field.lookup</i>	Launches the Look Up dialog box for a specified field.
<i>wdo_name.field.mark</i>	Marks the data field for error or attention. Sets <code>class="field mark"</code> on inputs.
<i>wdo_name.field.modify</i>	Switches to modified mode.
<i>wdo_name.next</i>	Calls the next batch of data from the client-side data cache.
<i>wdo_name.nextbatch</i>	Fetches a new batch of data from the server.
<i>wdo_name.prev</i>	Calls the previous batch of data from the client-side data cache.
<i>wdo_name.prevbatch</i>	Fetches a new batch of data from the server.
<i>wdo_name.reset</i>	Resets values and stays in update mode.
<i>wdo_name.save</i>	Saves the current row.
<i>wdo_name.field.set value</i>	Sets a data value in named field.
<i>wdo_name.undo</i>	Undoes all changes. (Applicable when autoconnect is not used.)
<i>wdo_name.update</i>	Switches to update mode.
<i>wdo_name.view</i>	Switches to view mode.

Examples

The following example shows multiple actions for the customer WDO:

```
app.apph.actions(
  ["customer.zip.disable"           // Disable customer.zip
    ,"customer.name.enable"         // Enable customer.name
    ,"customer.country.focus"       //Set focus to customer.country
  ]);
```

The following example shows how to get the customer state field screen value. If there is no screen value, then the underlying data value is retrieved:

```
apph.action('customerfullo.state.get');
```

The following example shows how to execute the dynamic lookup for a field:

```
apph.action('customerfullo.state.lookup');
```

The following example shows how to set the customer state field screen value. Note that the value is separated by the pipe (|) symbol:

```
apph.action('customerfullo.state.set|NH');
```

The following example shows how to display the customer state field, presumably after it was previously hidden by *wdo.field.hide*:

```
apph.action('customerfullo.state.show');
```

A.2 JavaScript for client-side functions and commands

Client-side functions can execute in response to SDO events and TreeView events.

A.2.1 SDO functions

You can create a function for a client-side SDO event by using the following naming convention:

`SDOName_event()`

For *event*, you must use one of the following:

Event	Function description
add	Executes after a record has been added.
copy	Executes after a record has been copied.
delete	Executes prior to a delete. Expects a return value.
dynLookup	Executes before return from a dynamic lookup.
update	Executes before validation.
validate	Executes before validation. Expects a return value.
view	Executes after data has been refreshed.

Example

The following example creates some customizations based on the value of `city` in the customer table:

```
// Sample Javascript include for Client-side SDO logic.

// Colors the city field yellow for Nashua, but otherwise red
function customerfullo_view(){
    window.document.all('customerfullo.city').style.backgroundColor=
        (apph.action('customerfullo.city.get')==Nashua?'yellow':'red');
}

// Sets some default values on add operation
function customerfullo_add(){
    apph.action('customerfullo.name.set|default name');
    apph.action('customerfullo.address.set|default address');
    apph.action('customerfullo.city.set|Nashua');
}

// same for copy as for add
function customerfullo_copy(){
    customerfullo_add();
}

// Special treatment of Nashua as value for city field both for validate and
update

var city_a;
var city_b; // being passed from validate to update so it must have outside
scope

function customerfullo_validate(){
    city_a=apph.action('customerfullo.city.old');
    city_b=document.all('customerfullo.city').value;
    if(city_a==Nashua&&city_b!=Nashua){
        alert('I will not let you do that because Nashua is my favorite city!');
        return false;
    }
    return true;    // Tells whether the operation was legal...
}

function customerfullo_update(){
    if(city_b!=Nashua)
        apph.action('customerfullo.city.set|'+city_b+'=bad city');
}

// Refuse to delete records where city is set to Nashua
function customerfullo_delete(){
    if(apph.action('customerfullo.city.get')==Nashua){
        alert('I will not allow people from Nashua to go away');
        return false; // Tells whether the operation was legal...
    }
    return true;
}
```


A.2.2 TreeView commands and functions

You can use the JavaScript API to control the following client-side processing:

- [Clearing the current TreeView](#)
- [Identifying TreeView data structure](#)
- [Capturing TreeView events](#)

Clearing the current TreeView

You can clear data from the current TreeView with the following statement:

```
dyntree.clear(parent_row_number)
```

This command clears data from nodes below the parent node specified by *parent_row_number*. If *parent_row_number* is omitted, the entire TreeView is cleared.

Identifying TreeView data structure

Use the following statement to get information about the data structure of the current TreeView:

```
dyntree.element
```

The value of *element* can be any of the following:

JavaScript reference	Returns DOM reference to
<code>TABLE.rows[<i>row_number</i>].cells[0]</code>	The TD node of the specified row of the HTML table for the TreeView.
<code>struct['<i>node_name</i>']</code>	An associative array of properties defined for the specified node.
<code>currentopen</code>	The TR element for the currently open node.
<code>selected</code>	The TR element for the currently selected node.
<code>currentwdo</code>	The value of <code>hdata</code> for the currently open source data object

Capturing TreeView events

You can create a function, `tvEvent`, in your client-side JavaScript to capture TreeView events. Use the following syntax:

`tvEvent(event)`

For *event*, you must use one of the following:

Event	TreeView state
launch	Waiting for server to return container.
fetch	Waiting for server to return child nodes.
extract	Extract program node is selected.
empty	Node with no container is selected.
view	Data node is selected.
update	Data node is in update mode.
add	Data is being added.
modify	Data is being edited.

Example

The following example illustrates the use of some of the TreeView API calls:

```
// department.js - Filter viewer for Department treeview

function filterButton(){
    dyntree.clear(1); // Clear the nodes underneath the department node (0)

    // Setting the filter criteria
    document.form['_departmentfullo._filter'].value=
        'deptname > '+apph.logic.widgetValue('fifrom').split(',')[0]
        +'| deptname < '+apph.logic.widgetValue('fito').split(',')[0];

    apph.actions([
        'server.departmentfullo.filter',      // filter command
        'server.wbo.0.department.dyntree',  // treeview data fetch command
        'wbo.submit']);                     // submit the request
}

// Capture treeview events
function tvEvent(status){
    window.main.status='tvEvent='+status;
    viewerMode(status!='modify' && status!='add');
}

// Custom function to set state of viewer input fields
var vmode=true;
function viewerMode(mode){
    if(vmode==mode) return;
    vmode=mode;
    var inputs=document.forms['_departmentfilterv'];
    for(var i=0;i<inputs.length;i++){
        if(vmode){
            inputs[i].removeAttribute('disabled');
        } else {
            inputs[i].setAttribute('disabled','disabled');
        }
    }
}
```


Index

A

ActiveX controls 4–8

add event A–13

agent. *See* WebSpeed agent.

alternative visualizations 4–8

Apache Web server 3–26

app actions A–3

AppBuilder 1–7

application data 2–12

application development

- ActiveX controls 4–8
- alternative visualizations 4–8
- buttons 4–9
- client-side features 1–4
- custom UI events 4–8
- customizing 5–1, 6–1
- deployment 4–15
- dialog boxes 4–7
- features 1–2
- help 5–7
- I/O blocking 4–9

- menus 4–9
- overview 1–2, 4–2
- resizing browse columns 4–13
- running an application 4–3
- server-side features 1–3
- starting a session 3–39
- tool support 1–5
- unsupported objects 4–13
- Windows applications 4–7

architecture

- client-side 2–11
- Progress Dynamics 2–1
- server-side 2–2

audience ix

auditing 4–13

AutoComments property 4–11

B

bold typeface

- as typographical convention xi

broker. *See* WebSpeed broker

browse columns 4–13

- resizing 4–13

browser. *See* Web browser.

- business logic 6–1
 - context management 6–3
 - I/O blocking 6–3
 - invoking from a client 6–2
 - invoking JavaScript APIs 6–3
 - output parameters 6–3
 - server-side 6–2

- buttons 4–9

C

- Can Run Locally option 3–9, 3–11

- Cascading Style Sheets 1–4, 5–8
 - applying themes 5–14
 - class selector 5–10
 - contextual selectors 5–11
 - customizing 5–12
 - customizing with 5–12
 - default files 5–9
 - HTML element selector 5–10
 - ID selectors 5–11
 - links in default HTML file 5–4
 - rules 5–9
 - StyleSheetFile attribute 5–12

- CLASS attribute 5–10

- class selector 5–10

- client logic API 6–5, 6–6
 - logic object 6–5
 - object qualification 6–6
 - qualification with browse keyword 6–6
 - qualified instance names 6–6

- client-side
 - architecture 2–11
 - data management 2–12
 - logic 4–8, 6–1

- Comments
 - restrictions 4–11

- comments
 - indicator in browser 4–15

- Container Builder 1–7

- container data
 - refreshing in browser 4–13

- contextual selectors 5–11

- copy event A–13

- CSS. *See* Cascading Style Sheets

- custom UI events 4–8

D

- database connections 3–36

- default HTML file 1–4, 5–2
 - CSS links 5–4
 - JavaScript links 5–5
 - layouts 5–3
 - page title 5–5
 - SCRIPT elements 5–5
 - TITLE element 5–5
 - XHTML conformance 5–5

- delete event A–13

- deployment 4–15

- development mode 3–33

- development restrictions 4–12

- DHTML. *See* Dynamic HTML.

- dialog boxes 4–7

- dlg actions A–3

- Document Type Declaration (DTD) 5–5

- dyn attribute 3–21

- Dynamic Call Wrapper 1–3

- Dynamic HTML 2–11, 5–1

- dynamic lookups
 - development requirements 4–9

dynamic object area 5–3

Dynamics Images utility 5–21

Dynamics. *See* Progress Dynamics.

dynLookup event A–13

DynSports

- logical service values 3–9
- physical service values 3–11
- starting 3–8

E

encoding attribute 5–6

Error messages

- displaying descriptions xii

extension mapping 3–22

- Apache Web server 3–27
- Microsoft IIS 3–23

G

getPropertyList function 6–4

H

handling Web requests 2–4

help

- application 5–7
- Progress messages xii

hidden frame 5–3

HTML 5–2

HTML element selector 5–10

I

.icf extension 3–22, 3–25

ICFDB

- starting 3–8

ICFDBn 3–4, 3–14

ICFDevAs 3–4

ICFSESSTYPE 3–17, 3–31

ICFWS session type 2–8, 3–2, 3–15

- adding a database to 3–8
- copying 3–15
- default settings 3–3
- extending 3–15
- modifying default session 3–8
- removing a service 3–14
- testing configuration 3–37

ID attribute 5–11

ID selectors 5–11

IIS. *See* Microsoft IIS

images 1–4, 5–21

info actions A–4

Internet Explorer 1–9

Internet Information Services. *See* Microsoft IIS.

I/O blocking 4–9, 6–3

Italic typeface

- as typographical convention xi

J

JavaScript

- action syntax A–2
- actions A–2
- adding functions 5–19
- API 1–4, A–1
- app actions A–3

- client-side events A-12
- customizing with 5-16
- dlg actions A-3
- extending object types 5-17
- info actions A-4
- JavaScriptFile attribute 5-16
- JavaScriptObject attribute 5-18
- links 5-5
- main actions A-6
- object files 1-4, 5-15
- objects 5-14
- prg actions A-6
- server actions A-6
- summary 5-15
- tool actions A-7
- util actions A-8
- wbo actions A-9
- wdo actions A-10

JavaScriptFile attribute 5-16

K

Keystrokes xi

L

layouts 5-3

LOG_PATH 3-35

LOG_TYPES 3-35

logging setting 3-34

logic object 6-5

login 2-8

look and feel
for Web applications 4-7

lookup files
filtering choices in browser 4-14

lookups
development requirements 4-9

M

main actions A-6

managers 3-5

Manual, organization of x

menu bar 5-3

menus 4-9

message area 5-3

Messages

- displaying descriptions xii
- Info object 2-13

messages

- controlling display of 5-4

Microsoft IIS 3-22

Monospaced typeface
as typographical convention xi

Mozilla 1-9

O

Object Generator 1-5

object qualification 6-6

online help 5-7

Open Client DataObject 1-3

P

page title 5-5

pcPropertyList parameter 6-3, 6-4

pcPropertyValues parameter 6-3

.pf file 3-31

Physical Service Control window 3-10

platform support

plSessionOnly parameter 6–3, 6–4

popups
 in SmartViewerObjects 4–11

prg actions A–6

production mode 3–33

Progress Dynamics
 architecture 2–1
 Broker and agents. *See* WebSpeed broker
 or WebSpeed agents.
 customizing Web applications 5–1, 6–1
 handling Web requests 2–4
 JavaScript API Reference A–1
 Web applications 4–1
 Web Test Page 3–37
 WebSpeed 2–2

Progress Explorer 3–17, 3–28

PROPATH 3–31

property sheets 1–6

R

refreshing container data 4–13

Request Manager 1–3, 2–6, 2–8, 2–10

required managers 3–5

Run Locally option 3–9, 3–11

ryabout.r 5–7

ryapp.css 5–4, 5–9

ryhelp.r 5–7

ryinithtml.js 2–5

rylogic.js 6–5

rymain.css 5–4, 5–9, 5–10, 5–11

S

SBO. *See* SmartBusinessObject

SCRIPT elements 5–5

Security Manager 2–8

server actions A–6

server-side
 architecture 2–2
 business logic 6–2

sessions
 add a database to 3–8
 ICFDBn 3–4
 ICFDevAs 3–4
 ICFSESSTYPE 3–31
 ICFWS 3–2
 ICFWS settings 3–3
 removing a service from 3–14
 required managers 3–5
 services 3–4
 Session ID 2–7, 2–8
 Session Manager 2–7, 2–8
 Session Type Control 3–3
 settings 3–2

setClientAction procedure 6–3

setPropertyList function 6–3

SmartBusinessObject
 development requirements 4–10

SmartViewerObject
 popups in 4–11

svrStartupParam 3–31

STATE_AWARE_ENABLED 3–35

static files 1–4, 3–20

StyleSheetFile attribute 5–12

T

themes 5–14

TITLE element 5–5

tool actions A–7

tool bar 5–3

tool support 1–5

Toolbar and Menu Designer 1–7

TreeViews

- API commands and functions A–15

- development considerations 4–11

- display and behavior 4–14

typographical conventions xi

U

UI data 2–12

UI Manager. *See* User Interface Manager

Unicode UTF-8 support 5–6

unqualified names 6–6

unsupported objects 4–13

update event A–13

URL syntax 2–4, 3–38, 4–6, 5–14

user considerations

- browser behavior 4–13

User Interface Manager 1–3, 2–6, 2–7, 2–9, 2–10

util actions A–8

utilities 1–4

V

validate event A–13

view event A–13

virtual directories

- Apache Web server 3–26

- Microsoft IIS 3–22

W

wbo actions A–9

WBO. *See* WebBusinessObject

wdo actions A–10

WDO. *See* WebDataObject

Web applications

- look and feel 4–7

- running 4–3

- setting up for 3–1

- See also* application development.

Web browser

- behavior 4–13

- requests from 2–4

- support 1–9

Web server

- Apache 3–26

- configuring 3–19

- Microsoft IIS 3–22

- testing 3–36

Web Test Page 1–8

WebBusinessObject 1–5

WebDataObject 1–5, 2–12

WebSpeed -----2-2
 environment variables 3-35
 request 2-5
 URL syntax 2-4, 3-38, 4-6

WebSpeed agent
 application modes 3-33
 context management 2-7
 logging setting 3-34
 managers 2-6
 setting up 3-27
 startup parameters 3-17, 3-31

WebSpeed broker 2-5
 in default HTML file 3-21

 logging setting 3-31
 properties 3-30
 setting up 3-27
 specifying 3-21

Write to Config option 3-9, 3-11

.wsc extension 3-24

wsdynamics1 2-5, 3-17, 3-29

X

XHTML conformance 5-5

