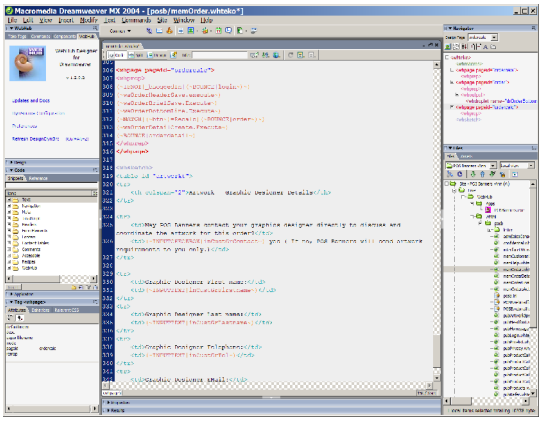
WebHub Designer for Dreamweaver 8



User Guide

v 2.00

Table of Contents

1. Introduction

Welcome

Welcome to the next level of Web Application development !

The *WebHub Designer for Dreamweaver 8* is an integrated suite of *Dreamweaver Extensions* that enable a developer/designer to visually and interactively create and maintain the page design and content for dynamic WebHub Applications. (WebHub is a product created and marketed by HREF Tools Corp. - see www.href.com).

This guide will help you to install the *WebHub Designer* into Dreamweaver, it will explain configuration options, and it will clearly describe all new features available for the design and management of WebHub whteko files.

After reading the application programming instructions in this guide, you will be able to construct an entire set of pages for a WebHub Application that runs dynamically, or export some or all of these pages to static html files.

Best practices (including adherence to current Web Standards) are encouraged and referred to and/or detailed throughout this document.

This User Guide includes a section ("Definitions") which clarifies important and often misunderstood terms. It is coming up in a couple of pages! Please remember to refer to this section whenever you notice unfamiliar words and phrases used in this Guide.

Target Audience

The content of this user guide is aimed at both *Web Page Designers* and *WebHub Program Developers.* Your role need only be one of these, but of course it may be both. Network administrators may also find this guide useful in so far as folder layout and configuration issues are concerned.

As regards the *Web Page Designer* role, this guide assumes at least a basic familiarity with some core features of Dreamweaver. Though not essential, this familiarity is helpful. The core issues are: creating a Dreamweaver Site, opening and saving files locally, putting and getting remote files via ftp or accessing files via a LAN, and switching between Code-view and Design-view.

All levels of Web Page Designer experience are catered for. You will find this guide useful whether you have had experience with static html pages only, or dynamic pages such as PHP, Cold Fusion, ASP or ASP.NET, WebHub Pages using Stage 1 Syntax, or WebHub pages that use Stage 2 syntax (which includes the `<whteko_tag>` content structure).

If you are a developer who compiles WebHub programs, then this guide presents all the information required to compile and configure a WebHub Program such that it can be used as a Dynamic Content Source by Dreamweaver.

Even if you are a highly experienced Dreamweaver user, *Web Page Designer* or *WebHub Program Developer*, there is a great detail of content in this guide (including sections that appear to cover only basic material) that is worthwhile reading. The contained best-practices suggestions are worth taking the time to explore, as they are structured to save you design and development time and also to streamline future maintenance of your web application projects.

Key Features

Each of the features available in the *WebHub Designer* will be described in detail, but first here is a very brief overview.

The *WebHub Designer* provides a Dreamweaver user with complete management of WebHub ***.whteko*** documents and offers the following key features:

- support for a new dynamic-document-type called 'WebHub v2' that has a .whteko extension.
- support for WebHub Stage 2.14 syntax, with full color highlighting.
- insertion and editing of all WebHub <whteko> tags.
- the ability to have any number of html pages in one whteko file.
- easy selection of a current *design page* within a multi-page whteko file.
- support for a new design <!DOCTYPE named 'whteko' which enables full validation of whteko file contents, automatically including xhtml validation of all contained html code.
- a *Navigator* Panel for overviewing and easily traversing code within a whteko file.
- a Panel-Group named 'WebHub' containing 4 panels named as follows:

  WebHub - providing easy access to extension-updates, documentation, configuration, preferences and WebHub Application Refresh.

  Teko Tags - listing all WebHub <whteko_tags> (with quick code insertion feature).

  Commands - listing all built-in WebHub Commands (with quick code insertion feature).

  Components - listing the most commonly used WebHub components Properties and Methods (with quick code insertion feature).

- a WebHub Expression Editor for inserting and editing WebHub expressions in html code.
- an integrated dialog for setting Preferences.

Features specific to visual design of page content are:

- an integrated dialog for setting one or more WebHub dynamic-content sources.
- a Command (or keypress) to refresh a local or remote WebHub application.
- rapid visual design of multilingual content

and most importantly

- real-time, unaltered, complete code rendering of dynamic-content for the selected design page in Dreamweaver's Design-view.

Note: These last four visual design features require that a properly configured WebHub Application be available over local or remote http. See **2. Planning** - **System Requirements** and also **3. Installation & Configuration** - **Installing a WebHub Application Server**.

Definitions

There are a number of words, phrases, terms, abbreviations and placeholders that are used repeatedly throughout this document. The most important of these are presented in this section with *simplified* definitions.

As you are reading this user guide and encounter a term that is unfamiliar, we recommend returning to this section for an explanation. You may however skip over this entire section and proceed straight to **2. Planning** – **System Requirements**.

General Terms

Project

That aspect of a web site for which you have a design and/or maintenance responsibility.

Life cycle

The iterative process (within a project) of design-refine-test-deploy.

Development computer

The local computer that Dreamweaver is installed on.

Production Server

This is an environment in which a live web application runs in its final form as used by end-users as contrasted with testers and developers. In the case of a public web application, the production server is easily identified because it runs the public domain. In this user guide, we will assume that the Dreamweaver developer does not want to work directly with the Production Server but rather with a staging and/or practice server. A production server always exists at the point of deploying a web site, although it may not exist in early development phases.

Practice Server

A Practice Server is an environment that is configured to simulate a Production Server. It is intentionally isolated to avoid problems with affecting the Production Server during development and subsequent design changes. Depending on your chosen design topology, the Practice Server may be located on the Development computer or on a separate computer. In very rare cases (e.g. when high risk is not an issue, and the developer owns the web site, and hardware is in short supply), the Practice Server environment may exist on the same computer as the Production Server.

HTTP Server Software

Software implementing the server side of the http(s) protocol, the client side being most often a web browser, also known as web server software. The most commonly installed HTTP Server software on Windows is named Microsoft IIS (Internet Information Server). There are other HTTP Server software packages available, such as Apache, Sambar and WebSite, and these are often used when the Microsoft operating system license does not include a free copy of IIS.

Practice HTTP Server Software

HTTP Server software that is running within your Practice Server environment.
See also: **Practice Server**

web site

This is a user experience that results from visiting an http host which, behind the scenes, can be creating content for you in myriad ways! The definitive source of any web site is to be found within the HTTP Server Software which responds to the http request for the domain. Multiple domain names and/or IP#s can lead to a single web site, and, a single domain name can lead to multiple web sites, all depending on configuration choices and creative use of dynamic web applications. As an example, the address http://www.google.com leads to one web site when used by someone in the United States, and to a different web site when used by someone in Australia, or China. Both web sites "are google" yet they give a different experience (different search results) to the site visitor.

Web Site

In this guide, when used with a capital **W** and capital **S** as in a **Web Site** rather than **web site**, this refers to a single Web Site as defined within HTTP Server software. In the case of IIS6, this would be one of the Web Sites configurable through Internet Information Services (IIS) Manager and is found under *Internet Information Services - ComputerName (local computer) – Web Sites*.

Web Site Document Root

This is an on-disk folder that is configured within HTTP Server software for a particular Web Site. It maps to the document root of a public, intranet or local domain name.
e.g. c:\InetPub\wwwroot is the default document root folder for IIS. For all public sites, the Web Site document root should contain, at a minimum, a file named robots.txt (refer to http://robotstxt.org). Static sites generally contain a file named index.htm, index.html, default.htm, default.html or default.asp in the document root folder. Dynamic sites do not necessarily have any *.htm, *.html or *.asp files in their document root folder.

Virtual Directory

Virtual Directories are sometimes called Virtual Roots, Aliases or Directory Mappings. Whatever the name, this is a directory mapping for HTTP Server software to map a **folder in the URL** of an incoming request to a **folder location on-disk**.

e.g. HTTP Server software may be configured such that a virtual directory named `Project1Abbrev` maps to the on-disk folder c:\Projects\`Project1Abbrev`. Then, any incoming request that has a URL starting with http://localhost/Project1Abbrev will cause the HTTP Server to locate a resource contained in the URL to be relative to that mapped folder i.e. relative to c:\Projects\`Project1Abbrev`. So an incoming request with a URL such as http://localhost/Project1Abbrev/default.html would return the contents of the on-disk file c:\Projects\`Project1Abbrev`\default.html.

The purpose of virtual directories is to separate certain parts of a web site from the document root folder. This flexibility is required when the Production Server directory layout and permissions are controlled by someone other than the *Dreamweaver Developer*. By using virtual directories, the developer is assured that last-minute search-and-replace adjustments are not required at the moment of deployment to the Production Server.

localhost

This is a universal predefined host for use within all internet oriented software. The host name **localhost** always maps to **127.0.0.1** (this is the loopback IP address and points to the computer that the request is made on). When making http requests with a browser on your local computer and using localhost as the host in the URL i.e. http://localhost/...., the response to your request will be handled by the Web Site within your HTTP Server software that is configured to serve the IP address 127.0.0.1. Literally, **localhost** means 'the host running locally'.

Dreamweaver Developer

A Web Page Designer who uses Dreamweaver to design web pages. In the context of this user guide, "Dreamweaver Developer" also includes WebHub Programmers who use Dreamweaver to edit *.whteko files in Code-view, but rarely use Design-view.

WebHub Program Developer

This is the individual, team or organization that designs and compiles WebHub Application EXEs using a CodeGear Delphi compiler with WebHub Object Pascal components installed.

Sandbox

This is a a local copy of files for development purposes. This term comes from the world of version control, where one checks out (or "updates") a set of files from the central repository into a local sandbox. When the developer decides that certain file modifications are final and worthwhile, he or she checks in (or "commits") those changes back to the central repository. Although a sandbox can be on a network drive, the idea is that it is under the control of a single developer, and it is thought of as "local" to the developer.

WebHub Terms

Teko

Teko is an Esperanto word that means **briefcase** and is used metaphorically in WebHub to indicate a container of one or more things. Those things may be declarations and content for WebHub pages, part pages, shorthand macros, human language translations and comments.. The letters "teko" are used as part of the file extension for WebHub page content (`.whteko`) and also as part of the main WebHub xml tag that contains child page-content tags `<whteko>` .

WebHub Connection Layer

This is the software that connects the HTTP Server software with running WebHub Applications and optimizes performance and scalability. See **Appendix C**.

WebHub Application

In a simplistic sense, a WebHub Application consists of:

- a compiled and running WebHub Application EXE (WebHubApp EXE)
- an **XML file** that contains configuration and default settings for this EXE
- one of more `whteko` files that contain the WebHub page content

The WebHub Application creates dynamic content responses in real time for page requests on a web site.

WebHub Application EXE

This is a compiled WebHub program with an EXE extension. WebHub Application EXEs can run under a logged in user as an application or they can run as a service. On development computers they are normally run as applications, and on production servers they are usually run as services. On a production server, WebHub Application EXEs are started when the computer boots, and run 24x7. A WebHubApp EXE loads its associated **XML** configuration file and `whteko` file content when it first runs, and also each time it is asked to Refresh.

WebHub XML configuration file

This is an initialization/configuration file containing settings that are used by its associated WebHub Application EXE. The settings in this file are not hard coded into the EXE such

that an Administrator or Dreamweaver user can quickly make a change to settings and then have these new values loaded by sending a Refresh request to the WebHubApp EXE. Importantly, this file contains the list of **whteko** files that will be loaded and used by the WebHubApp.

whteko file

A whteko file is a xml compliant plain-text file that contains an optional but recommended preliminary WebHub DOCTYPE tag, followed by a hierarchy of special xml tags which are called, as a group, **<whteko_tags>**. These tags contain declarations and content for WebHub pages, part pages, shorthand macros, human language translations and comments.

<whteko_tags>

These are an hierarchical set of xml tags that are used to organize and define WebHub page content, droplets, macros, translations, documentation and html sketches. They are used exclusively inside whteko files and their names always begin with **<wh**. They were introduced into WebHub during 2004 (v2.032 to v2.037) and are officially defined in WebHub Syntax Stage 2.14.

WebHub Command

A WebHub Command is text within the html of a WebHub page that has the format **(~**RESERVEDCOMMANDNAME|parameters**~)**. The surrounding **(~ ~)** characters are called parentils, and these delimiters are used for all WebHub Expressions. For built-in WebHub Commands, the RESERVEDCOMMANDNAME is a reserved word that WebHub uses to activate a specific feature.

WebHub Expression

A WebHub Expression is any sequence of WebHub Commands and nested expressions that are surrounded by parentils. The parentils delimiters **(~** and **~)** are used to distinguish a dynamic WebHub Expression from the surrounding html and are also used for nesting expressions within expressions.

WebHub Expressions are used within HTML/XHTML markup to:

1. dynamically insert content into a web page while it is being generated by a WebHub Application (WebHubApp), and
2. interact directly with a WebHubApp's logic and data.

WebHub URL

This is a dynamic URL that, when sent as part of a request, returns page content from a running WebHub Application. The URL format (in long-form) is:

http(s)://domain/runnerpath/runner?AppID:PageID[:Session:parameters]

A WebHub URL always leads to a single AppID whether directly (where the AppID is within the URL) or indirectly (where the AppID is mapped to by filter in the HTTP Server Software, or is configured on the server).

If using a filter to remap incoming requests, WebHub URLs may appear in short form:

http(s)://domain/PageID[:Session:parameters]

HREF Tools Corp. markets such a filter; see www.streamcatcher.com. Short URLs are recommended for public sites where search engine placement is a priority.

For a more detailed explanation see **6. Application Programming** - **Designing Dynamic**

WebHub Page Content - WebHub URLs.

AppID

This is a WebHub **Application Identifier**. It is a configured identifier associated with a WebHubApp EXE. AppIDs tend to be from 2 to 8 letters and always start with a letter.
On a given computer, each AppID is associated with one XML file, which in turn defines the list of whteko files that the WebHubApp EXE will load. The WebHub Connection Layer uses the AppID, among other aspects of a WebHub URL, to route the incoming request to the appropriate running EXE. Note that the EXE could be started more than once, with all instances running the same AppID.

See also: **WebHub URL.**

PageID

This is a WebHub Page-Identifier i.e. the **name** of a WebHub page. WebHub pages are declared in `whteko` files using the `<whpage..>` tag.

A WebHub page may or may not show the same content each time it is requested, depending on its design. A WebHub page may include parts which are tailored (or removed) based on information known about the site visitor. Some examples are: translating text to the site visitor's lingvo, rotating advertising content, showing members-only content, and displaying links for web robots but not humans. It may be helpful to think of a WebHub page as a "template", although it differs from Dreamweaver templates in that there is never a need to "ripple" the template through existing content because the WebHub template creates the content directly, not indirectly.

See also: **WebHub URL.**

Session

This is a site visitor **Session Identifier**, used by a WebHubApp to save and load state. WebHub session identifiers are always numeric and the number of digits is configurable. Often referred to as a **session number**. See also: **WebHub URL**

WebHub URL Command

This is an optional string appended to a WebHub URL after the session number. All aspects of the WebHubApp (including any WebHub Expressions within whteko files) can access the value of the WebHub URL Command, and WebHubApp code and/or whteko file code may use it to fine tune operation and logic.
See also: **WebHub URL**

Abbreviations

The following abbreviations are used throughout this document.

Local Site Root (Dreamweaver Site Root)

This is the **Local root folder** for a **Dreamweaver Site.** It is the local on-disk folder under which all Web Site related files are stored.

Access from the Dreamweaver main menu is as follows:

Site | **Manage Site** [Edit] existing site or add [New] site
Advanced tab , *Category*: **Local Info**, *Field*: Local Root Folder

Preview URL

This is the URL that Dreamweaver sends to a browser when you use the **File** | **Preview in Browser** feature. This feature requires that you define a Testing Server for your Dreamweaver site. Dreamweaver calculates the Preview URL from the Testing Server's "URL Prefix" plus the relative path to the file being previewed. (The relative path is the path relative to the local site root).

WebHubApp

This is an abbreviation for a running WebHub Application
Importantly, when used, it is implied that you also have HTTP Server software and the WebHub Communication Layer running on the same machine.

WebHubApp EXE

This is short for WebHub Application EXE.

Place Holders

The following placeholders are used within many of the examples throughout this document.

`MyClient`

This represents a folder name for a client and is only used when designing projects for more than one client.

`Project1Name`

This represents a folder name for your project and this folder is referred to as a project-folder. We recommend keeping the name as short as possible, using between 3 and 30 characters and avoiding spaces.

`Project1Abbrev`

This is a carefully chosen abbreviation for a project that is used as a folder name for organizing static resources. This should be as short as possible and cannot be equal to any WebHub AppID running on the same web site, nor any PageID within those AppIDs. Hint: If you have trouble coming up with a unique abbreviation, add “res” to your abbreviation, where “**res**” stands for “resources”. For example if you have an AppID of 'demo' and for some reason this is also the most useful abbreviation of your project, then you could use '**demores**' as the value for *`Project1Abbrev`*. Alternatively you could prefix the AppID with “**wr**” (short for web resources) to obtain '**wrdemo**', or suffix with "**design**" to obtain "**demodesign**".

2. Planning

System Requirements

The *WebHub Designer* software requires:

i) Dreamweaver 8 (installed on a Windows platform)

ii) a WebHub Application Server (running either locally or remotely) **

The WebHub Application Server provides dynamic content for Dreamweaver's Design view.

A WebHub Application Server consists of the following components:

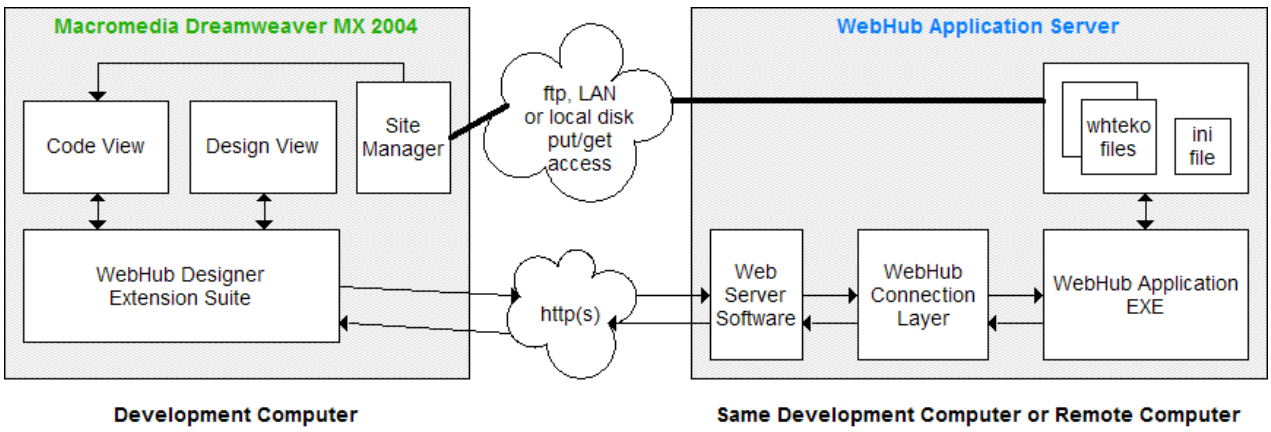
- *HTTP Server Software* (eg. Microsoft's Internet Information Server IIS5, 5.1 or 6)
- the *WebHub Connection Layer* (a WebHub runner and Hub system)
- one or more running *WebHub Applications*

** If you are using a remotely hosted WebHub Application Server, you do not need to install one on your local development computer.

With the *WebHub Designer* installed and with access to a WebHub Application Server, then in addition to the familiar interaction with local and remote files, Dreamweaver's Design-view is able to display dynamic content (generated by a running WebHub Application EXE).

The basic system architecture for operation of the *WebHub Designer* is shown in [Figure 2-1].

Figure 2-1 WebHub Designer System Architecture



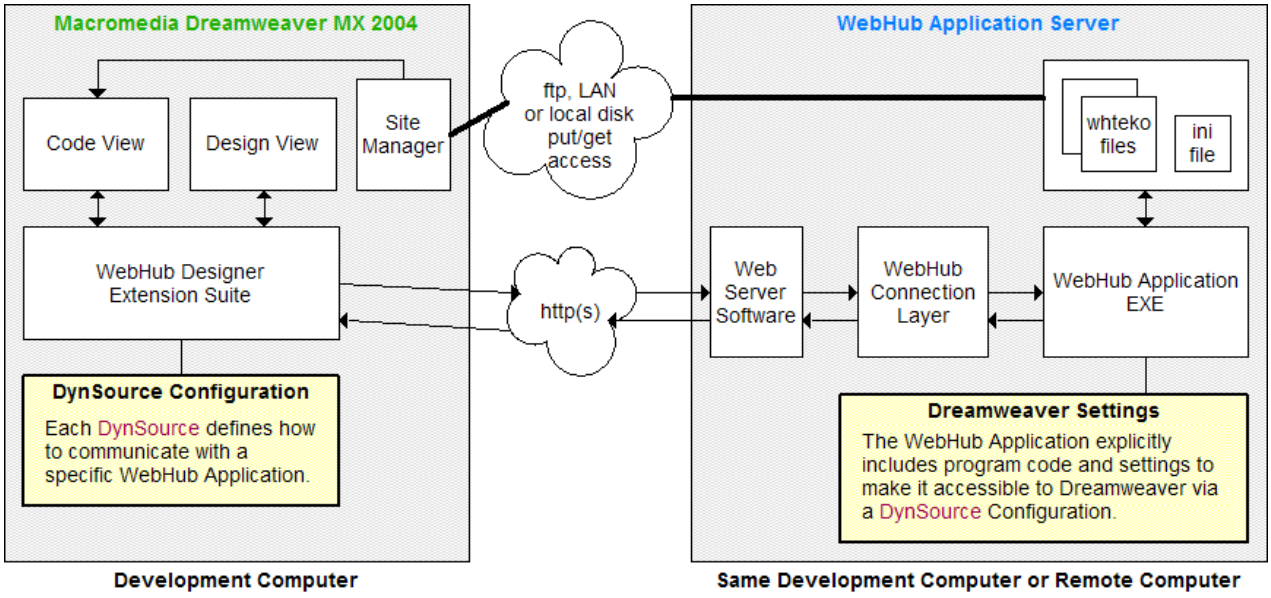
A running WebHub Application EXE is considered to be a **Dynamic Content Source**.

Identifying your Dynamic Content Source

Dynamic content for Design-view in Dreamweaver is provided by a **Dynamic Content Source** which is just a WebHub Application Server running one or more WebHub Application EXEs.

Dreamweaver accesses a running WebHub Application EXE via a user-configured *DynSource* (This is a *Dynamic Content Source alias*).

Figure 2-2 A DynSource definition enables Dreamweaver to communicate with WebHub



Note: Each **whteko** file that you work with has an opening xml tag in which you can specify the use of particular *DynSource* for that file. You may work with one or more WebHub Application EXEs (therefore one or more *DynSources*) within a given project.

In order to configure a *DynSource* that defines how to communicate with a specific WebHub Application EXE, *certain information* about the WebHubApp and its settings will be required.

This is fully explained in **3. Installation and Configuration:** – **Configuration** – **Configuring Dynamic Content Sources**.

If you are personally responsible for making the WebHub Application EXE then you can skip ahead to **Choosing your Design Environment Topology,** otherwise you should plan to contact the WebHub Program Developer responsible for the WebHubApp in order to obtain the following required information:

Required DynSource Configuration Details

URL for access to the WebHubApp EXE =
PageID of the *remote-design* page =
PageID of the *remote-refresh* page =
PageID of the *remote-preview* page =
Security *session-number* for access =
WebHub Expression-Prefix to use =

An example reply from a WebHub Program Developer responsible for a particular WebHubApp might be:

DynSource Configuration Details

URL for access to the WebHubApp EXE	=	**http://localhost/scripts/runisa.dll?MyApplD**
PageID of the *remote-design* page	=	**remotedesign**
PageID of the r*emote-refresh* page	=	**remoterefresh**
PageID of the *remote-preview* page	=	**remotepreview**
Security *session-number* for access	=	**1112**
WebHub Expression-Prefix to use	=	**(~**

WebHub Expression-Prefix Note

For backwards compatibility with WebHub v0.850 to v2.031, the WebHub Designer extension supports the `%=` expression prefix and the `=%` expression suffix, which were used in these versions. For example, syntax highlighting on *.whteko files works with `%=` ... `=%`, and dynamic content can be calculated for Design-view. We recommend that you change older projects to use `(~ ~)` as soon as possible, because some features only work with `(~ ~)` affixes, and moreover, the code will be more human-readable when `(~ ~)` parentils are used.

A search-and-replace utility named FuzRegex is available at http://www.fuzregex.com, along with scripts for converting %=...=% expressions to the corresponding (~...~) expressions.

Next is a detailed discussion of how to choose your design environment topology.

Choosing your design environment topology

Before using Dreamweaver to design WebHub pages, it is essential that you choose and/or identify the topology that will be used to design, share, access and transfer files.

The usual constraints for a design topology are budget, convenience, size and nature of production site users, skill level of developers and team size coupled with team member proximity.

We will consider 7 typical scenarios ranging from a low budget minimalist configuration through to a modest setup for the design and maintenance of a high profile web site.

The scenarios are:

A.	Small, personal or non-profit project hosted at home office	(1 developer, 1 computer)
B.	Single developer handles all roles	(1 developer, 2 computers)
C.	Single developer handles all roles for high visibility website	(1 developer, 3 computers)
D.	Typical small web team	(2 developers, 3 computers)
E.	Typical small web team handling high visibility site	(2 developers, 4 computers)
F.	Larger team handling high visibility sites	(3+ developers, 5 computers)
G.	Larger team handling high visibility sites with a testing team	(3+ developers, 6 computers)

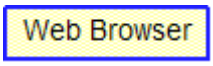
Note that if there are 2 or more Dreamweaver developers working on the same application, we highly recommend using some form of version control.

These scenarios will be presented with diagrams and then summarized in a table to compare and contrast development team size and nature, the number and type of computers required and the main benefits.

Diagram Legend - Design environment topology

In the scenario diagrams below, common symbols are used to represent each of the major components of software and hardware required. Each symbol indicates a combination of software and files with an implied base configuration as follows:

This represents Dreamweaver 8. A Dreamweaver Site is always defined for the project, with a local root folder (Local Site Root) set to point to an appropriate folder containing all website design files, and a Dreamweaver Testing Server configured to allow file and http access to a Practice Server. Remote Info (Remote Site Server) configuration details are dependent on the scenario and are noted below each diagram.



This represents a Web Browser and is used with the Practice Server. A developer would use the browser to directly test a WebHub Application by simulating an end-user. When implementing the **File** | **Preview in Browser** feature, Dreamweaver uses the Testing Server configuration (and a relative path) to send an appropriate URL to the Web Browser so that the current WebHub page being designed can be previewed.

This represents a WebHub Application Server (HTTP Server software, WebHub Connection Layer and running WebHub Application EXE) that is used as a **Practice Server** with 3 main purposes:

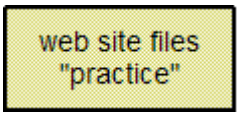
i) it is used by Dreamweaver (utilizing a *DynSource* configuration) to obtain **dynamic content** for inclusion in Design-view

ii) it is used by Dreamweaver (utilizing a Testing Server configuration) for **previewing** WebHub pages

iii) it is used by a Developer (using a browser) to **test a WebHub Application** in an environment that is isolated from the Production Server

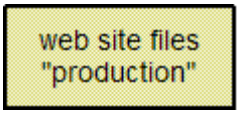
This represents a WebHub Application Server (HTTP Server software, WebHub Connection Layer and running WebHub Application EXE) that is used as a **Staging Server** to fully simulate a Production Server Environment. It is used to test and fully evaluate a complete working version of a WebHub Application prior to moving that application into production. Files are generally transferred to a Staging Server from the Practice Server and this is done outside of Dreamweaver.

This represents a WebHub Application Server in its complete final deployed form, and indicates a live to the public (or live intranet) web site. We do **not** recommend using the Dreamweaver File | Preview in Browser feature with a Production Server. Files are generally transferred to a Production Server from Staging Server (if one exists), or from the Practice Server, and this is done outside of Dreamweaver.

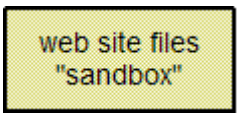


This represents web site design files (whteko, XML file, stylesheet, graphics, javascript files etc) that are used by a Practice Server environment.

This represents web site design files (whteko, XML file, stylesheet, graphics, javascript files etc) that are used by a Staging Server environment.



This represents web site design files (whteko, XML file, stylesheet, graphics, javascript files etc) that are used by a Production Server environment.



This represents web site design files (whteko, XML file, stylesheet, graphics, javascript files etc) that are used as a local sandbox design copy by a Dreamweaver developer. When changes are made in the sandbox area, then for testing and previewing the files are copied to a Practice Server environment.

Any additions or alterations to the base configuration implied by these common symbols, if applicable, are summarized below each scenario diagram.

Diagrams - Design environment topology

Figure 2-3 Scenario A - Small, personal or non-profit project hosted at home office

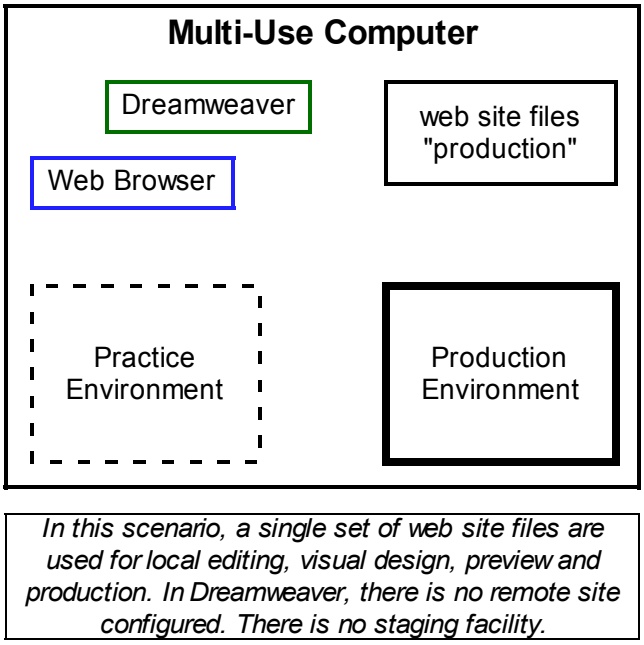


Figure 2-4 Scenario B - Single developer handles all roles

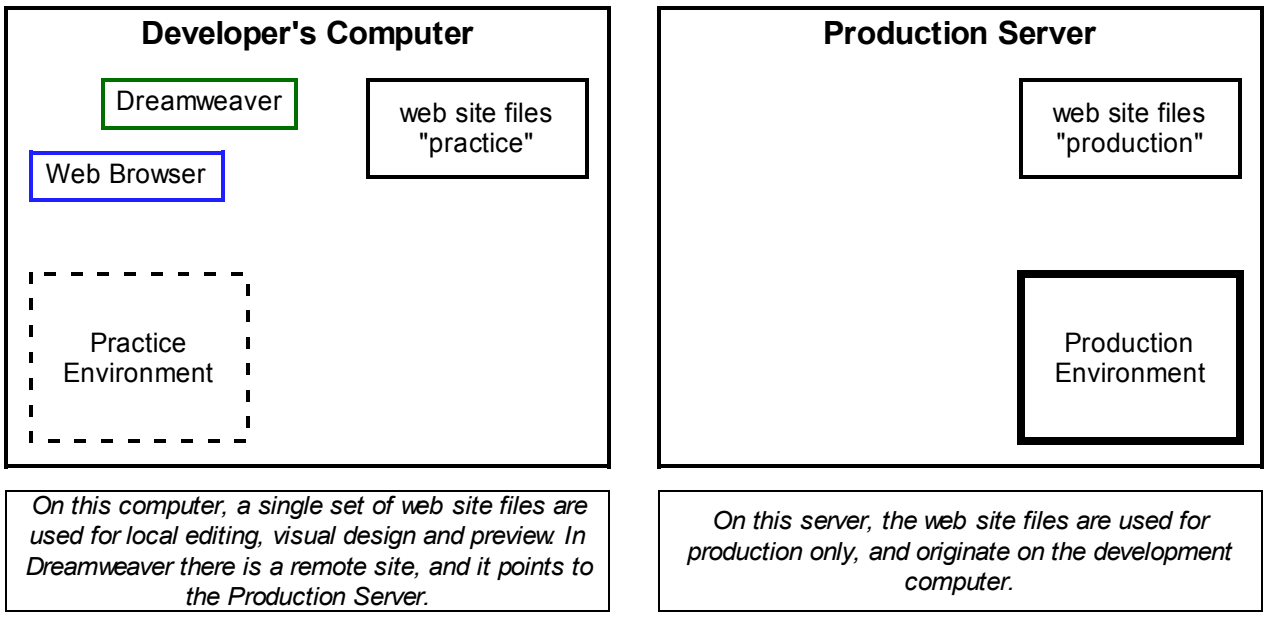


Figure 2-5 Scenario C - Single developer handles all roles for high visibility website

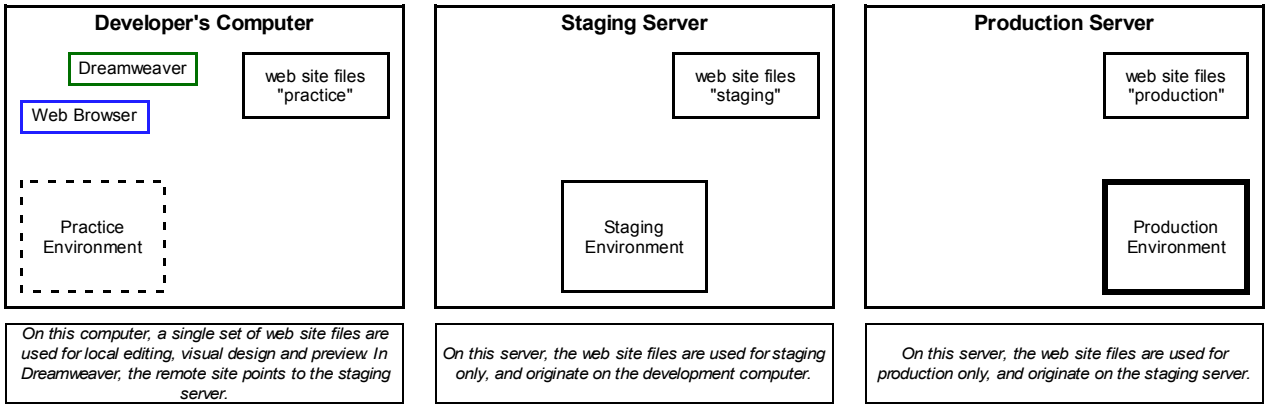


Figure 2-6 Scenario D - Typical small web team

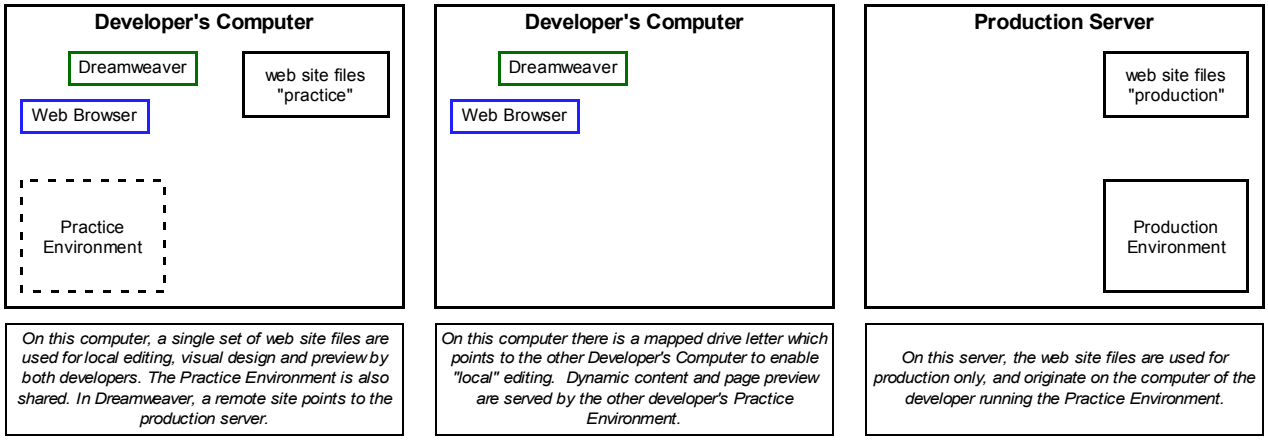


Figure 2-7 Scenario E - Typical small web team handling high visibility site

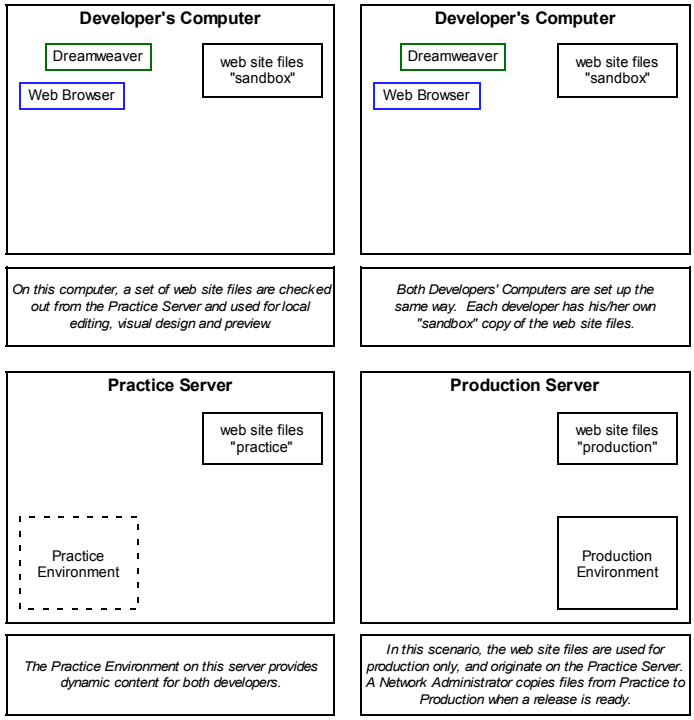


Figure 2-8 Scenario F - Larger team handling high visibility sites

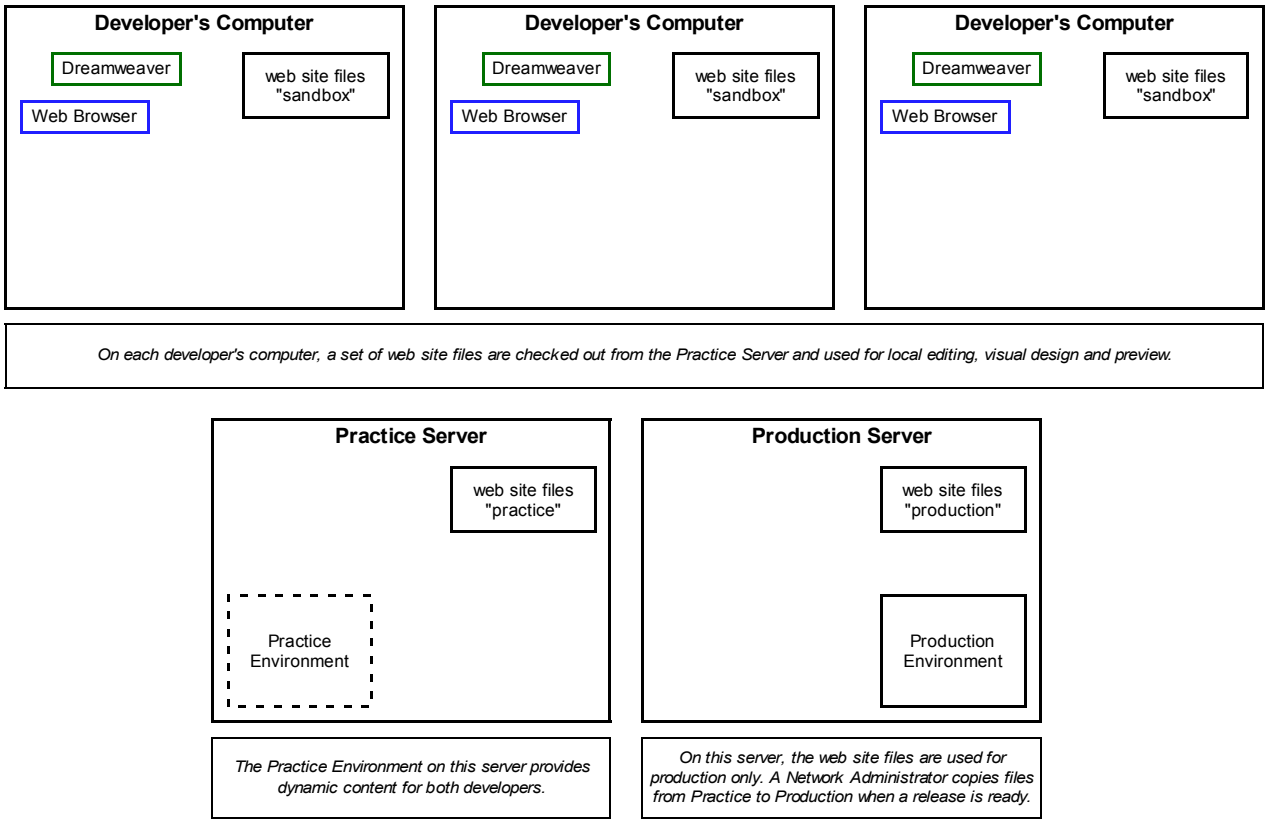
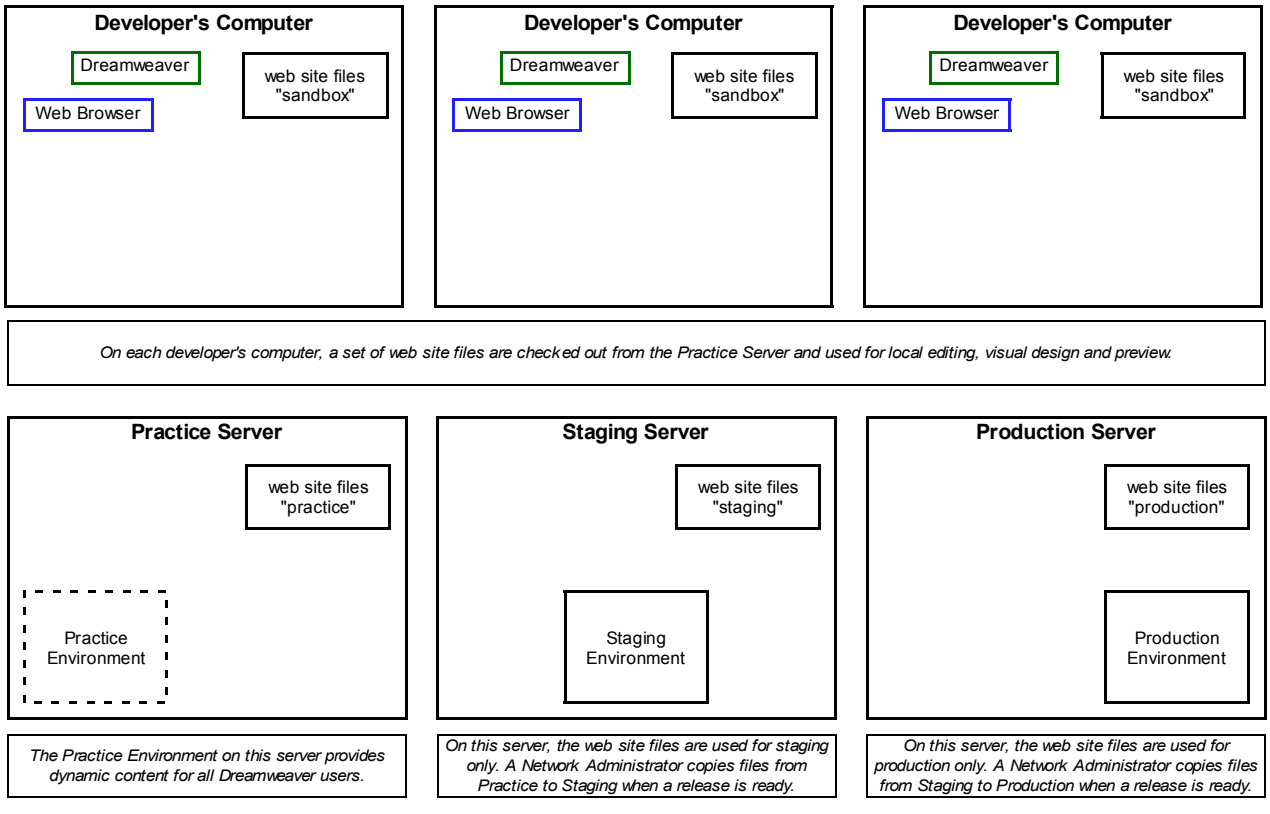


Figure 2-9 Scenario G - Larger team handling high visibility sites with a testing team



Comparison Table - Design environment topology

Each scenario suits a particular development team size and has benefits that flow from the number of computers available and also from the application server software that is running.

A comparison of the main attributes and benefits of each topology is shown in [Table 2-1].

Table 2-1 Comparison of topology development team, computers (# and type) and main benefits

	Scenario						
	A	**B**	**C**	**D**	**E**	**F**	**G**
Team							
# developers	1	1	1	2	2	~3+	~3+
Computers							
# development computers	1	1	1	2	2	~3+	~3+
# separate production servers	0	1	1	1	1	1	1
# separate practice servers	0	0	0	0	1	1	1
# separate staging servers	0	0	1	0	0	0	1
# computers - total	1	2	3	3	4	~5+	~6+
Benefit							
low cost (relative to # developers)	X			X			
convenient	X	X	X		X	X	X
safe with high traffic			X	X	X	X	X
safe with VIP users			X	X	X	X	X
beginner skill sufficient			X	X	X	X	X
Development							
suits one developer	X	X	X				
suits tiny team of developers				X	X		
suits any size development team, at one or more locations						X	X
supports separate testing team			X				X

Connectivity Choices - Design environment topology

No matter which design environment topology is used, whether it is one of the above typical scenarios or a variation, each Dreamweaver developer still needs to configure a Dreamweaver Site in order to access and view the WebHub Application design files. A Dreamweaver Site must always have a local root folder (Local Site Root) configured, and we are recommending that there is always a **Testing Server** configured so that the **File** | **Preview in Browser** feature is available. Configuring **Remote Info** for access to a Remote Server Site is **optional** and dependent on the topology used.

[Table 2-2] summarizes access methods (the most commonly used plus optional methods) that would apply to each of the typical scenarios above. The summary is presented for the main configuration areas of a Dreamweaver Site – **Local Design Files** (Local Info), **Remote Info** and **Testing Server**.

Table 2-2 Access methods available for a Dreamweaver Site for each topology scenario

Topology	***Scenario***						
	A	**B**	**C**	**D**	**E**	**F**	**G**
Team							
# developers	1	1	1	2	2	~3+	~3+
Computers							
# development computers	1	1	1	2	2	~3+	~3+
# separate production servers	0	1	1	1	1	1	1
# separate practice servers	0	0	0	0	1	1	1
# separate staging servers	0	0	1	0	0	0	1
Configuration Item / Access							
Local Design Files (Access)							
Local	X	X	X	X	X	X	X
Network				X	X	X	X
Testing Server (Access)							
None (testing server is optional)	X	X	X	X	X	X	X
FTP				X	X	X	X
Local/Network	X	X	X	X	X	X	X
Remote Info (Access)							
None	X						
FTP		X	X	X	X	X	X
Local/Network		X	X	X	X	X	X
SourceSafe Database				X	X	X	X
WebDAV				X	X	X	X
CVS				X	X	X	X

Dreamweaver's Viewpoint - Design environment topology

We will now discuss the design environment topology from the point of view of a Dreamweaver Developer.

Specifically we will consider how Dreamweaver sees and can can access files and other computers through the configuration of 3 primary file views.

In Dreamweaver these views are named **Local view**, **Remote view** and **Testing Server** (view). Sometimes Remote view is called Remote Site View.

Each of these views is configured within a Dreamweaver Site definition. Configuring a Dreamweaver Site for the design of WebHub Application pages is essential. As a minimum, you must define the *local root folder* of your site. We generally recommend configuring a Testing Server and also configuring a Remote Site -- whenever possible.

As to file locations accessed by these views, in section **6. Application Programming** - **Designing Dynamic WebHub Page Content** - **Folder Layout and File Location for WebHub Applications** there is a best-practices suggestion for locating files on the development and practice / production servers. The more carefully you and the Network Administrator structure folder layout and file locations, the easier it will be for everyone to coordinate file transfer within your topology.

Each of the Dreamweaver file views is now separately discussed.

Local view and Local Files

Local files (i.e. design files, namely `whteko`, css, javascript, image files and the xml file) may be located either

i) on your development computer or

ii) on a LAN-connected computer using a mapped drive.

(The mapped drive allows Dreamweaver to see files as local even though they are located on another computer in a network)

These **Local files** are shown in Dreamweaver's **Local view** in the Files panel.

For this view to work, you need to correctly configure the Local Site Root for the active Dreamweaver Site. When defining a Dreamweaver Site, there is a field on the Advanced tab (Local Info category) labeled **Local root folder** (referred to in this guide as Local Site Root or Dreamweaver Site Root). You will set this to a particular parent folder such that it contains all of your design files and sub-folders. This is more fully explained in later examples.

Remote view and Remote Site

For static page development, Dreamweaver's **remote site** (remote info) feature is used to **transfer files to and from a production server.**

For WebHub page development, however, the **remote site** feature may also used to **transfer files to and from a Practice Server** (depending on your topology). If the Practice Server (which runs HTTP Server Software, the WebHub Connection Layer and a WebHub Application) is on your development computer then you will ***not*** define a remote site. Even if your Practice Server is on a separate machine, configuring a Dreamweaver **remote site** is only possible when connecting via one of Dreamweaver's supported protocols. The supported protocols for WebHub Applications are: LAN, FTP, SourceSafe or WebDAV.

If you are using an unsupported (as far as Dreamweaver is concerned) version control system such as CVS, you will not be able to define a remote site, rather you would transfer files using the version control system itself (e.g. “CVS commit”)..

To configure a **Remote Site** to connect to a Practice Server, you would use the **Remote Info** category on the **Advanced Tab** in the Site Definition dialog. The Remote Folder (or Host Directory if using ftp) is configured to access the Practice Site Root.

Once a Remote Site has been defined, you will be able to view the Practice Server's folder structure and its file details by choosing **Remote view** on the Dreamweaver Files panel.

Testing Server

A Testing Server configuration is only required if you want to use Dreamweaver's **File** | **Preview in Browser** feature. Using this feature is recommended.

Furthermore, we recommend configuring the Testing Server to access the Practice Server, i.e. the one pointed to by the *DynSource* specified in the **`<whteko_tag>`** of the file containing the page being previewed.

There are two situations in which using a Testing Server is extremely inconvenient, or even impossible. One case is when a single project contains files which reference two or more DynSources. This is inconvenient, because you would have to keep changing the Testing Server definition, depending on which file you were working on. The second case is when security or other rules within the dynamic content of the page make it impossible to Preview that page without either first logging in or stepping through the web application in a normal fashion. The solution, for both cases, is to ignore the File | Preview in Browser feature and just use a web browser for testing by typing in the URL yourself.

See also: www.url2image.com for testing site designs on various platforms.

3. Installation & Configuration

Installation

This section covers installation of the *WebHub Designer* Extension into Macromedia Dreamweaver 8, and also installation of a WebHub Application Server. If you are designing against a pre-existing Practice Server environment, you do not need to install a WebHub Application Server on your local development computer so just follow the installation instructions for the *WebHub Designer* and then skip ahead to **Configuration**.

Installing the WebHub Designer Extension

To install the *WebHub Designer*, complete the following steps.

1. **Close** Macromedia Dreamweaver 8 if it is running.

2. **If, and only if**, a prior version of the *WebHub Designer* has been installed then

    i) Tidy your Recycle Bin (restore any files that you really did not mean to delete) then Empty it. This step is suggested because uninstalling the WebHub Designer will place a considerable number of files into your Recycle Bin (around 100 or so) thus making it all the harder to search for previously deleted files that you may have wanted to restore but had not quite got around to rescuing.

    ii) **Start** Macromedia Extension Manager.

    iii) Ensure Dreamweaver *8* is showing in the drop-down control on the toolbar (if not then change the drop-down so that it does).

    iv) Uninstall the prior version of *WebHub Designer* by highlighting it, then choosing **File** | **Remove Extension**

    iv) **Close** Macromedia Extension Manager.

3. **Locate** the installation file **WebHubDWMgr.mxp** and **double-click** to trigger installation of the extension suite into Dreamweaver. This will automatically launch Macromedia Extension Manager.

4. Read the license agreement and if it is acceptable then click the **Accept Button.**

5. When the installation has finished, optionally read the displayed information describing the *WebHub Designer*, then **close** Macromedia Extension Manager.

6. Lastly, there is one final step to the installation that Macromedia's Extension Manager is unable to accomplish for you. This step will be completed by you selecting a menu command within Dreamweaver. So **launch** Dreamweaver and from its main menu choose **Command** | **WebHub Designer** | **Initialize .whteko file type**.

This completes the installation of the *WebHub Designer*. A successful installation is confirmed by being able to access all of the features described in the following sections.

See **Appendix A** for a complete list of all files created and changed by *WebHub Designer* during installation.

Installing a WebHub Application Server

The availability of a WebHub Application Server (*WebHub Application* running with the *WebHub Connection Layer* and HTTP *Server software*) enables the more advanced features of *WebHub Designer.* These advanced features are:

i) expanding WebHub Expressions (that are embedded in html pages) into dynamic content when switching to Design-view

ii) facilitating **Preview in Browser** for WebHub pages.

The *WebHub Application Server* may be run locally or remotely. If running locally, *HTTP Server software*, the *WebHub Connection Layer* and a custom WebHub Application must be installed on your local development computer.

Instructions for installing each prerequisite are included in the following Appendices:

Appendix B – WebServer Overview

Appendix C - Installing the WebHub Connection Layer

Appendix D - Compiling a WebHub Application EXE for use with Dreamweaver

Appendix E - Installing and Configuring a WebHub Application EXE for use with Dreamweaver

Please follow the instructions in those Appendices now, unless you are designing against a pre-existing Practice Server.

Next is a detailed look at configuration issues.

Configuration

Designing WebHub Applications with Dreamweaver requires a configuration that precisely matches your design environment topology (see **2. Planning – Choosing your design Environment Topology**).

There are two main areas of configuration required, one is configuring **Dynamic Content Source** access (and this configuration is unique to the *WebHub Designer*) and the other is configuring a Dreamweaver Site (and this is generic to Dreamweaver).

Configuring Dynamic Content Sources

Dynamic Content Sources are WebHub Applications (running most commonly in a Practice Server environment) that provide dynamic content for Design-view. An alias called a *DynSource* is configured for access to the *remote-design* and *remote-refresh* pages of a specific WebHubApp.

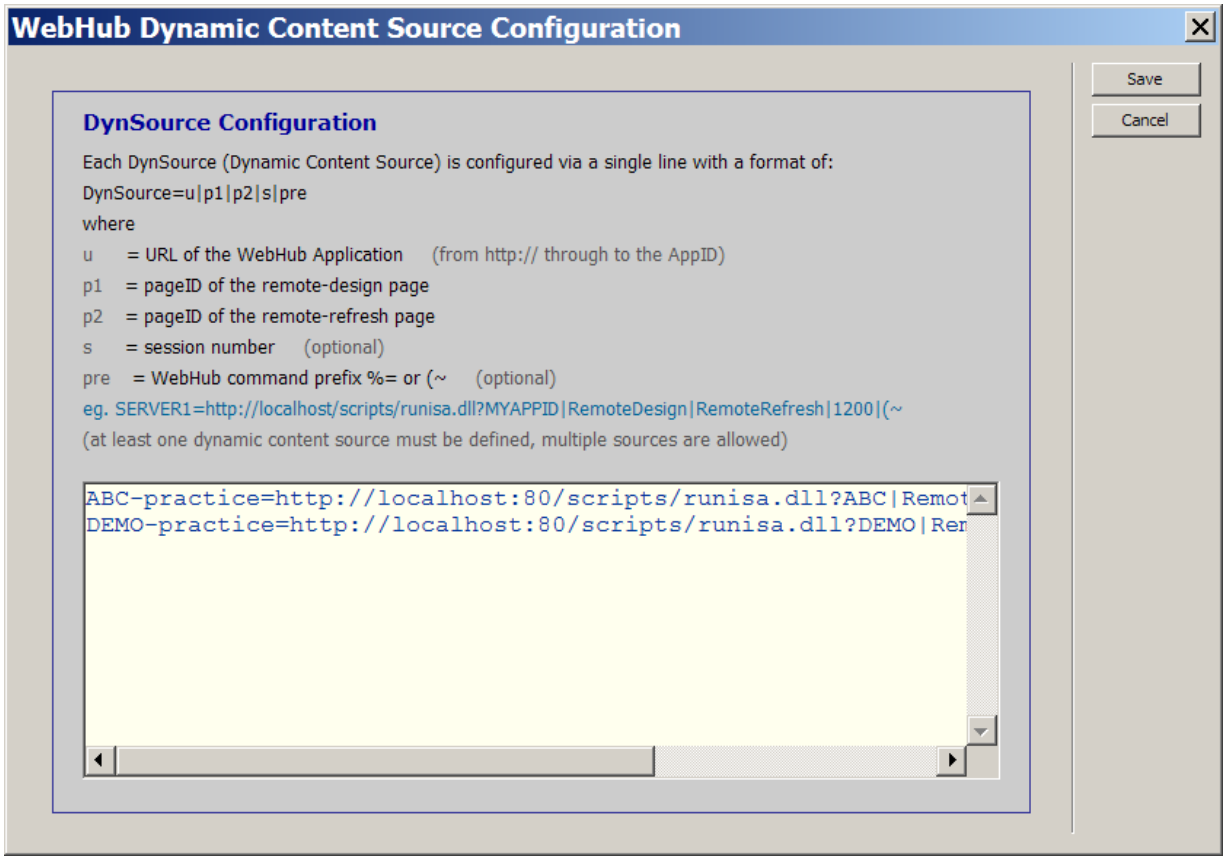
DynSource Configuration Access

Access to the *WebHub Dynamic-Content-Source Configuration* dialog is via:

WebHub panel-group: *WebHub panel* DynSource Configuration
or Main-Menu item: **Commands | WebHub Designer | DynSource Configuration**

This will launch the dialog named *WebHub Dynamic-Content-Source Configuration* [Figure 3-1].

Figure 3-1 WebHub Dynamic Content Source Configuration dialog.



DynSource Configuration Overview

This configuration dialog is used to define one or more *DynSource* entries, each pointing to the **remote-design** and **remote-refresh** pages of a specific WebHub Application. Access to these pages enables dynamic content for Design-view and the WebHubApp refresh feature.

Each *DynSource* is defined on its own separate single line.

Pressing the [Save] button writes your configuration settings to the WebHub Designer configuration file.

DynSource Configuration Example

```
abc-practice=http://localhost/scripts/runisa.dll?abc|RemoteDesign|RemoteRefresh|
1214|(~
```

would define a *DynSource* named **ABC-practice**. (The format of the right of = will be defined in a moment).

DynSource Configuration Syntax

The format of a DynSource configuration line is as follows:

`DynSource=u|p1|p2|s|pre` where

`u` = URL for access to the WebHub Application (WebHubApp) that will serve design content

`p1` = PageID of the *remote-design* page within the WebHubApp

`p2` = PageID of the *remote-refresh* page within the WebHubApp
`s` = Session-number configured (in the WebHubApp) to allow Dreamweaver access
`pre` = WebHub expression-prefix used by the WebHubApp, either `%=` or `(~`

e.g.
```
abc-practice=http://localhost/scripts/runisa.dll?ABC|RemoteDesign|RemoteRefresh|
1214|(~
```

DynSource:	`abc-practice`
WebApp URL:	`http://localhost/scripts/runisa.dll?abc`
PageID of *remote-design* page	`RemoteDesign`
PageID of *remote-refresh* page	`RemoteRefresh`
Session-number:	`1214`
WebHub command-prefix:	`(~`

Dreamweaver would request content for Design-view using this URL

http://localhost/scripts/runisa.dll?abc:RemoteDesign:1214

and would request a WebHubApp refresh using this URL

http://localhost/scripts/runisa.dll?abc:RemoteRefresh:1214

(Note: The **Refresh Design DynSrc** feature (or **Ctrl+Alt+Z)** instructs a WebHub Application to reload and process all of its associated whteko files and configuration file).

DynSource naming – best practices

A *DynSource* name can be any value you choose. A best-practices approach however would be to include the **AppID** of the WebHubApp being pointed to, and also to include an indication of the type of server environment that the WebHubApp is running in. The server types are Practice, Staging and Production.

The suggested approach is to start the *DynSource* name with the **AppID in lowercase**, and follow this with one of the lowercase words **practice**, **staging** or **production**. The most common use of a DynSource will be to specify access to a WebHubApp running on a Practice Server.

e.g. for a WebHubApp with an AppID of 'abc' that is running on a practice server, the recommended name for a *DynSource* would be *abc-practice.*

DynSource Usage

A *DynSource* is only ever used within a **whteko** file (a file containing WebHub page content). Dreamweaver determines which WebHub Application to use for dynamic content by reading the configuration settings for the particular *DynSource* that is used in a whteko file.

More precisely, in a whteko file, when a *DynSource* is used as the value of the designdynsrc attribute in the opening **<whteko_tag>**, then when switching to Design-view, Dreamweaver is able to request dynamic content from the WebHub Application pointed to by this *DynSource*.

Example.

In a **whteko** file named *MyFile.whteko*, you may have an opening **<whteko_tag>** that looks like this:

```
<whteko
  designdynsrc="abc-practice"      ← this attribute specifies the DynSource
  defaultlingvo="eng"
  designlingvo="eng"
  designpage="mypageid"
  designmode="visual">
```

Here, the attribute named designdynsrc has a value of **abc-practice**. This would be one of the *DynSources* that you configured.

When editing the *MyFile.whteko* file, each time you switch from Code-view to Design-view (and assuming that a change to the code has been made), a request is sent to the WebHub Application pointed to by the DynSource named abc-practice (via http) to obtain dynamic content for display.

Next is a detailed look at configuring a Dreamweaver Site for WebHub.

Creating and Configuring a Dreamweaver Site for WebHub

For convenience, we strongly recommend that one of the first things you do when working on a new project is to create the project's directory structure, and to define a Dreamweaver Site associated with the top level of that directory tree. Within the Dreamweaver Site definition, there are 3 main categories: Local Info, Remote Info, and Testing Server.

Local Info: A Local Root Folder is configured to act as a Local Site Root (Dreamweaver Site Root). All Site-Root-Relative static resources within a WebHub page are resolved by Dreamweaver's Design view to be relative to this Site Root. The Local Site Root must be above all images, style sheets and javascript files referenced by the project, otherwise Dreamweaver will not be able to locate them.

Remote Info: The purpose of configuring Remote Info is to enable Dreamweaver to help you with viewing, copying, renaming and deleting files on a Remote computer, whether that is a Practice Server or Production Server.

Testing Server: Configuring a Testing Server enables **File | Preview in Browser** to work for WebHub pages within your site. Here, a Dreamweaver Site is configured for access the *remote-preview* page of a WebHub Application running on the Practice Server.

The settings in a Dreamweaver Site configuration depend on your **design environment topology** and **also on your folder structure** for containing web site design files. Because there are a large number of combinations of possible configuration settings, we will restrict the discussion to 3 topologies (scenarios B,D and E from the section on **Planning**) and a specific best-practices folder layout. The folder layout for the project will be assumed to be:

```
c:\Clients                          ( umbrella folder for clients and clients' projects )
    \MyClient                       ( individual client folder )
        \Project1Name               ( project-folder = Dreamweaver site root )
            \Live                   ( folder to segregate files for production sever upload )
                \WebHub             ( parent folder for WebHub-Application-Server files )
                    \Apps           ( folder for WebHub Application EXEs )
                        Demo.exe
                    \Library        ( optional folder for DLLs and Delphi packages )
                    \whteko
                        \demo
                            WHApp_Config_demo.xml
                            default.whteko
                            contact.whteko
                    \WebRoot        (parent folder for static resource files)
                        \Project1Abbrev   (mapped to by a virtual directory named Project1Abbrev)
                            \css
                                mystyles.css
                            \images
                                image1.png
                                image2.png
                            \js
                                myscripts.js

            \Source                 (folder to segregate development only files)
                \Docs
                    specs.doc
                \whApps
                    \demo           (folder for Delphi source to Demos.exe)
                        demo.dpr
                        demo_*.pas
                        demo_*.dfm
```

This folder structure is discussed in detail in **6. Application Programming** – **Designing Dynamic WebHub Page Content** – **Folder Layout and File Location for WebHub Applications**, but for now just take it on face value as this layout enables specific examples to be presented for configuration settings.

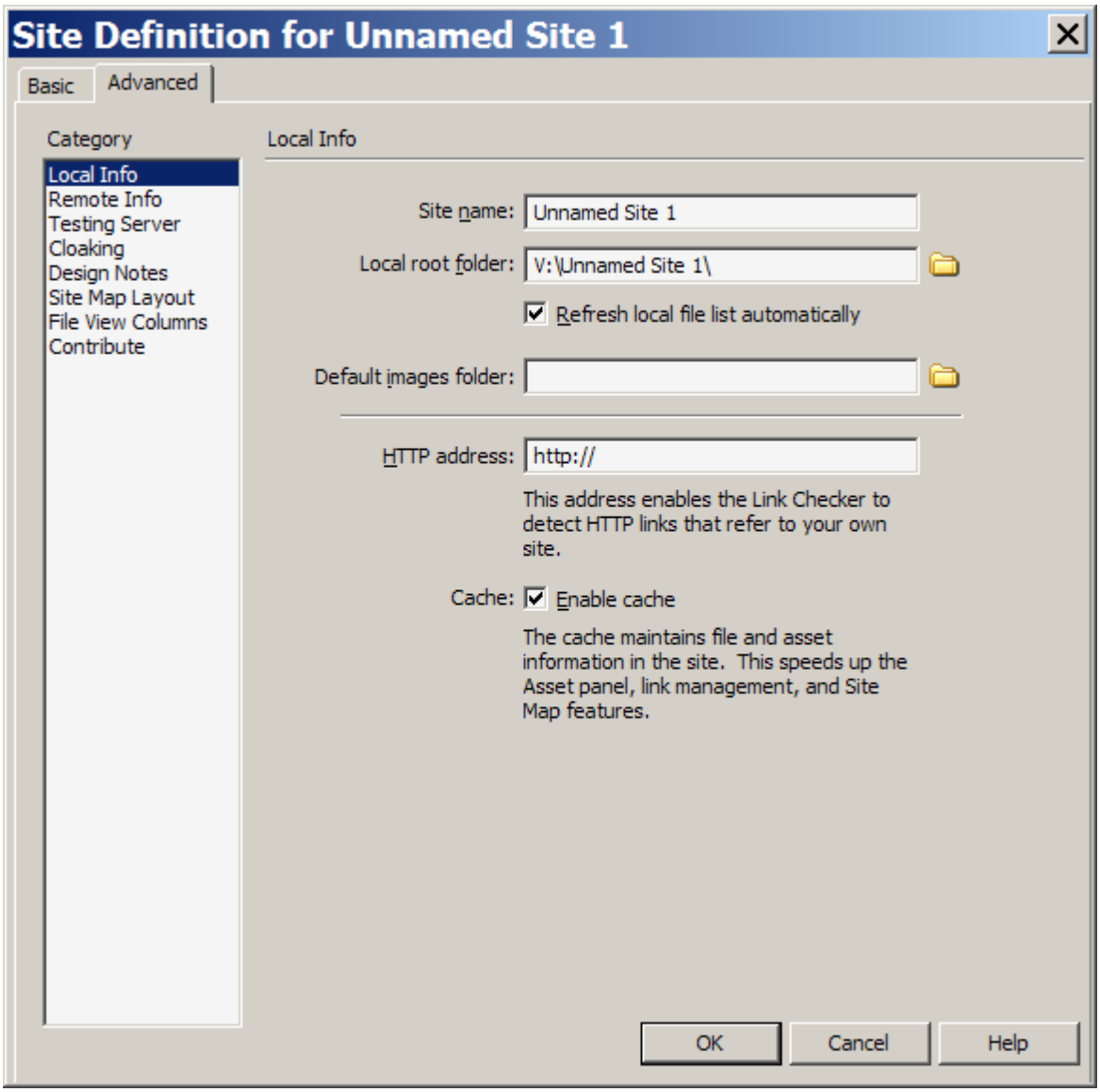
1. Start with Dreamweaver's Site Manager

To create a new Dreamweaver Site, the most straightforward way is to launch Dreamweaver's Site Manager form the Main Menu:

Site | **Manage Site** This will launch the Site Manager Dialog.

On this Dialog click the [New] button then choose **Site** from the Drop-down menu. You should then see the Advanced Tab of the Site Definition dialog as in [Figure 3-2]

Figure 3-2 Dreamweaver's Site Definition dialog – Advanced Tab – Local Info



2. Set the Local Info category options

The initial configuration Category to display on the Advanced Tab is **Local Info**. This is where you set the Local Site Root (Dreamweaver Site Root) for your project.

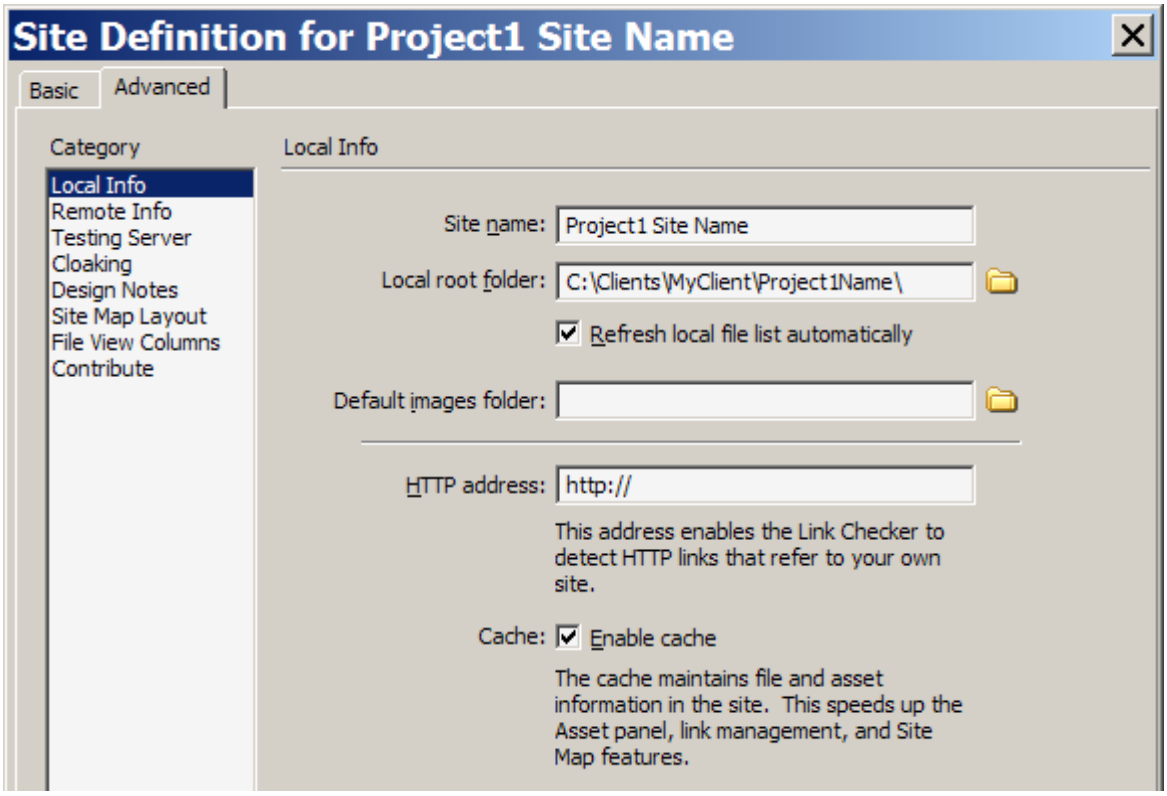
You will set this to a parent folder that contains all of your design files (namely **whteko**, css, javascript and image files and the **xml** file) and sub-folders. Depending on your topology, these files will either be a local sandbox copy, or will be the Practice Server's web site files directly.

As well as acting as the Root folder for Dreamweaver's **Local view** (for this site) in the Files Panel, all Site-Root-Relative static resources within a WebHub page are resolved by Dreamweaver's Design-view to be relative to this Local Site Root.

Remembering that placeholder names are in this font (*placeholder*), and using the sample folder layout, you would fill in the Local Info fields as follows:

Site Name:	*Project1 Site Name*
Local root folder:	c:\Clients*MyClient**Project1Name* (Local Site Root) ☑ Refresh local file list automatically
Default Images folder:	leave blank (as use is not recommended)
HTTP address:	*leave as http://* (this HTTP address URL for the Link Checker is not relevant when developing WebHub Applications).

Figure 3-3 Dreamweaver's Site Definition – Advanced Tab – Local Info



The value for Local root folder shown above assumed that the web site files were on the developer's local drive. If, due to your topology, these files were located on a LAN-connected computer, you would use a mapped drive e.g. v:*Project1Name* for the Local root folder (e.g. one of the developers in Scenario C would use a mapped drive).

3. Set the Remote Info category options

Overview

The **Remote Info** category is configured to access web design files, including both WebHubApp files and static resources, for a WebHub Application Server that is running on a remote computer.

Configuring access to a Remote Site is only necessary if you either have a remote Practice Server, or if your Practice Server is local and you also have a remote Staging Server or remote Production Server. Remote in this sense just means another computer that is available either by LAN connection or via one of the other Dreamweaver protocols - FTP, SourceSafe or WebDAV.

Among other things, the **Remote Info** configuration enables files to be transferred between your local copy of web site files and the files stored on a remote site.

For WebHub Application development, you would be transferring 2 distinct categories of files :

1. static resources (e.g. stylesheet files, javascript files and images) and
2. WebHubApp design files (`whteko` files and an XML file).

To this end it is easiest if you use the recommended folder layout on both your local and remote sites.

Note that in this layout, static resources are grouped under a folder named **WebRoot**, and WebHubApp design files are grouped under a folder named **WebHub**. Both these folders are sibling folders of a folder named **Live** that itself is a child of the main project folder *`Project1Name`*. Being able to see both distinct groups of files under a common parent folder is a significant advantage.

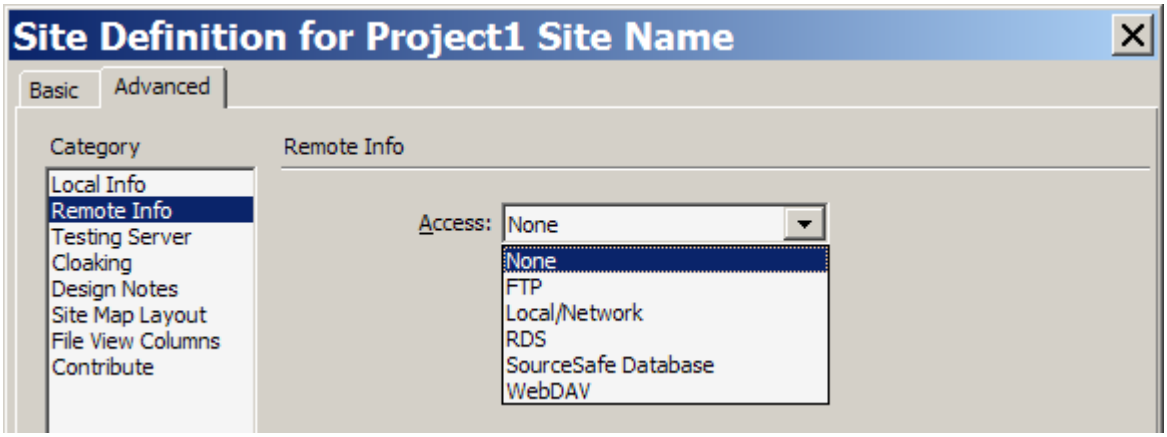
Following is a tabulated summary of the types of remote servers that are applicable to developers in each of the scenarios described previously in **2. Planning**.

Scenario	Developer(s)	Remote Info Server
A	Single developer	None
B	Single developer	Production Server
C	Single developer	Staging Server
D	Developer 1	Production server
D	Developer 2	None
E	Both developers	Practice Server
F	All developers	Practice Server
G	All developers	Practice Server

Specifics

To configure access to a remote site, on the Advanced Tab **click** the *Category* **Remote Info**. The first choice you will make is the method of access to the remote computer [Figure 3-4].

Figure 3-4 Dreamweaver's Site Definition – Advanced Tab – Remote Info

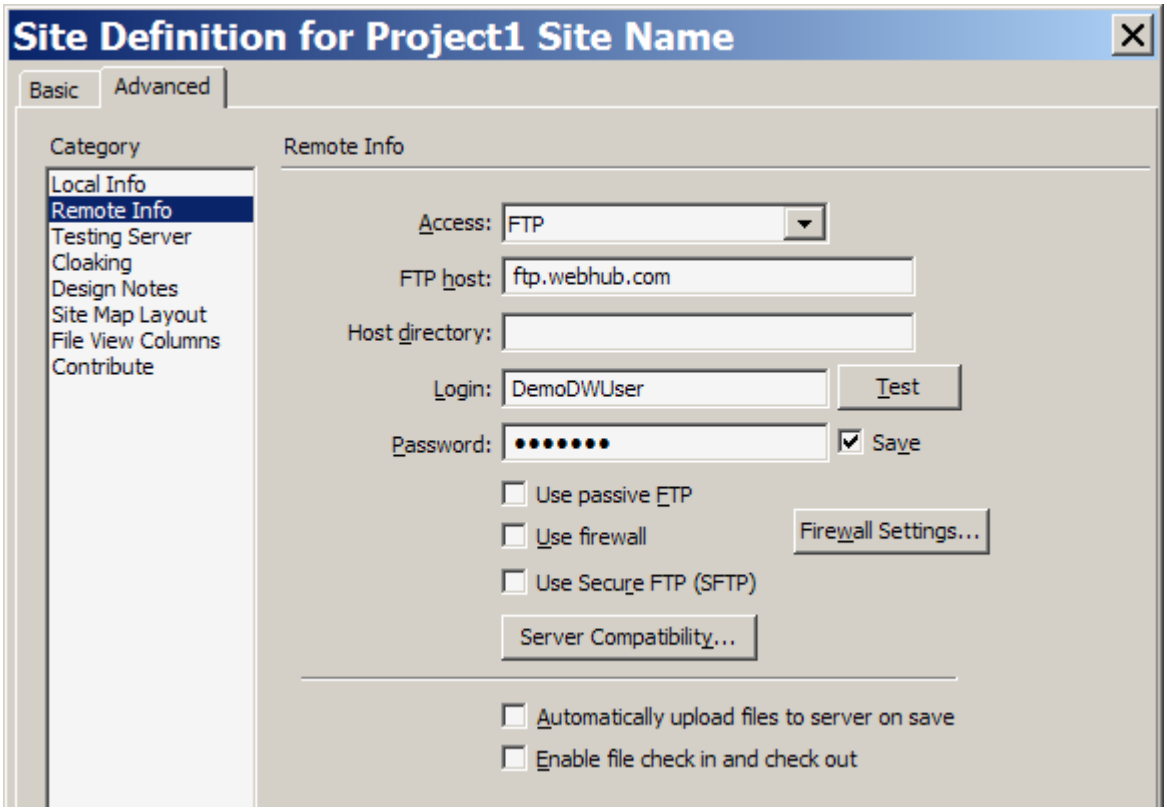


We will consider the access choices of FTP (the most common) and Local/Network.

Option 1: FTP Access: If you choose the access method to be **FTP**, then the Remote Info data entry fields are presented for connectivity to an ftp server running on the remote computer. Your access details will vary and you need to get them from the Network Administrator. We will show an example using the following sample settings:

Access: FTP
FTP host: ftp.webhub.com
Host directory: blank
Login: DemoDWUser
Password: ***** ☑ Save
Leave all other settings blank or configure them according to instructions from your Network Administrator.

Figure 3-5 Dreamweaver's Site Definition – Advanced Tab – Remote Info - FTP



For FTP access, the **Host Directory** is configured to access the Remote Site Root which is either the Practice, Staging or Production Site Root depending on your topology. The Remote Site Root needs to correspond with your Local Site Root as far as relative folder structures are concerned.

If the ftp server on the remote computer has been configured such that your Login and Password have a top root path that is the equivalent of your Local Site Root (e.g. equivalent to the local project folder of c:\Clients*MyClient\Project1Name)*, then as in the example you would leave the Host directory blank, because when logging in, Dreamweaver would already be in the correct directory.

When you have finished the necessary settings for ftp access, **Click** the **Test Button**. You should see the following reply (after a delay):

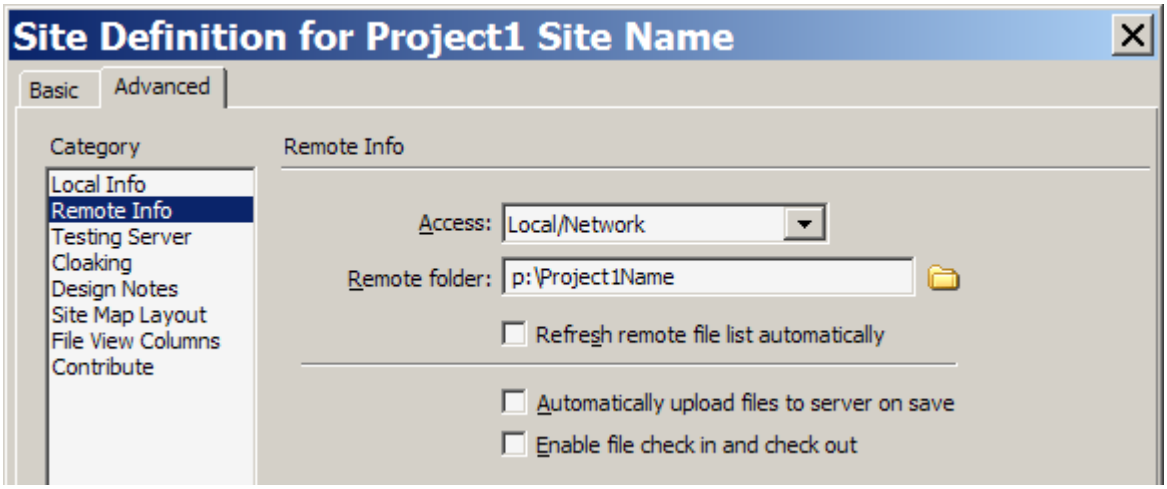
"*Macromedia Dreamweaver 8 connected to your Web Server successfully*."

Option 2: LAN Access: If you choose the access method to be **Local/Network**, then the Remote Info data entry fields are presented for local or LAN connectivity to the remote server environment. We will show an example using the following sample settings:

Access: Local/Network
Remote folder: p:\Project1Name (p: is a mapped drive)

For the remaining options, see Using Dreamweaver help under Setting Remote options for Local/Network access.

Figure 3-6 Dreamweaver's Site Definition – Advanced Tab – Remote Info – Local/Network



As before, the Remote Site Root needs to correspond with your Local Site Root as far as relative folder structures are concerned. For Local/Network access, the **Remote folder** is this Remote Site Root.

Note: the "Refresh remote file list" feature does NOT have anything to do with WebHub's refresh feature. You may leave it on or off, as you wish.

4. Set the Testing Server category options

Overview

Normally, the purpose of this dialog box is to specify a testing server which Dreamweaver would use to generate and display dynamic ASP, Cold Fusion or PHP content while you work in Design-view.

However when developing WebHub Applications, this aspect is handled by the *DynSource* configuration because a much greater flexibility in configuration options is required.

The only Testing Server feature used with WebHub is Dreamweaver's **File** | **Preview in Browser** feature, enabling you to preview WebHub page designs in one or more browsers.

If a Testing Server is not consciously configured, Dreamweaver uses the default options, namely Server Model: **None** and Access: **None.** In this case Dreamweaver sends a local file URL to the browser when previewing a file e.g. C:\Projects\MyProject\contact.whteko -- and this will never work for WebHub pages. This will never work because the WebHubApp EXE is not involved in calculating dynamic content, and because web browsers do not interpret `<whteko_tags>`. Therefore we use a unique technique to make Dreamweaver contact the WebHubApp EXE, and the EXE does the work of finding the design page within the whteko file, and calculating the page fully, dynamically. (See **4. Operation: Exploring Features** – **General Features** – **Preview in Browser**.)

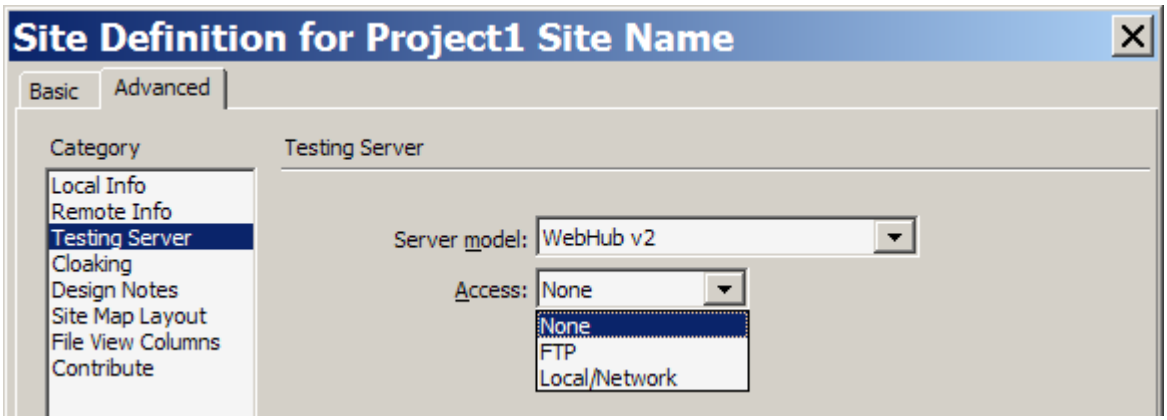
To enable Dreamweaver to preview WebHub pages, we configure Dreamweaver's Testing Server to:

1. save the file to the correct location in a Practice Server environment and
2. access the ***remote-preview*** **page** of a WebHub Application (running on the Practice Server) via a **URL Prefix** to which Dreamweaver appends the name of the file to Preview.

Specifics

To configure Testing Server options, on the Advanced Tab **click** the *Category* **Testing Server**. The first choice you will make is the Server Model – choose Server model: **WebHub v2**

Figure 3-7 Dreamweaver's Site Definition – Advanced Tab – Testing Server

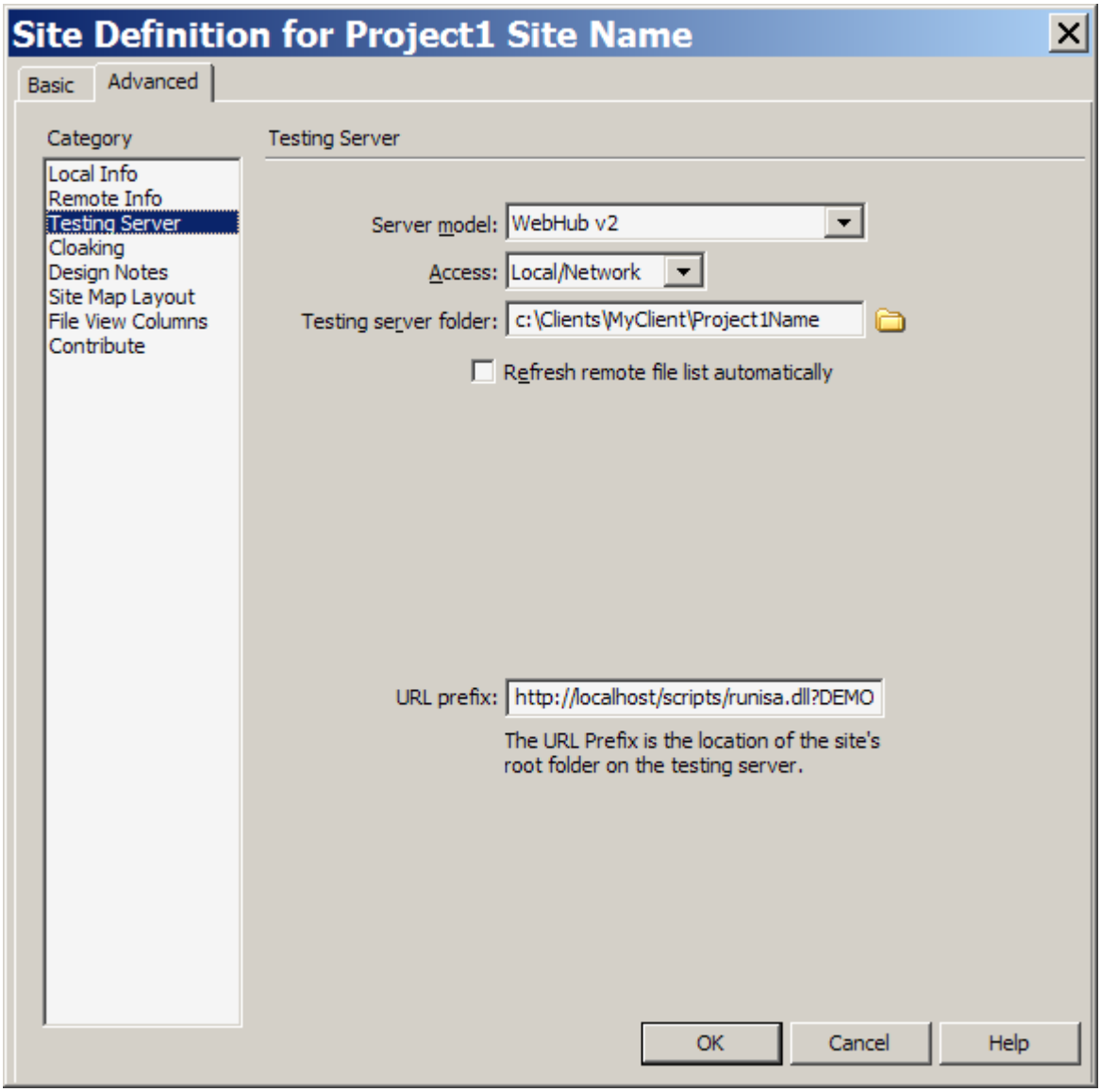


Next, choose the Access method that is appropriate to your topology (FTP or Local/Network).

For either access method, there are two key items to configure:

1. The Testing Server Folder (or Host Directory if using ftp)
2. The URL Prefix

Figure 3-8 Dreamweaver's Site Definition – Advanced Tab – Testing Server - Access



Testing Server Folder (or Host Directory if using FTP)
The Testing Server Folder (or Host Directory if using FTP) is configured to be the same as the Practice Server environment root. This is a folder on the Practice server that is the equivalent of your Local Site root.

Dreamweaver uses the Testing Server Folder to transfer the file being previewed. The design file is transferred such that it is relative to the Testing Server Folder so as to preserve the files location relative to the Local Site Root. This Folder is also used as the Root folder when viewing the Testing Server files in the File panel.

URL Prefix

This is the URL to the **remote-preview page** of the a WebHub Application running in your Practice Server environment. This is the *same WebHub Application* pointed to by the *DynSource* that is used in the file being previewed. If you use different DynSource values in whteko files within the same project, then you will need to ensure that the URL Prefix matches the DynSource being used. Mostly a single DynSource is used in the whteko files.

When you use the **File | Preview in Browser** feature, the relative path (i.e. relative to the Local Site Root) of your design file is added to this URL prefix before being sent to a browser to preview a page in the design file.

When previewing, Dreamweaver first copies the local design file to a location relative to the Testing Server Folder that is the equivalent of its location relative to the Local Site Root. Then the URL prefix + relative path to the design file is sent to the WebHub Application.

The above example has a URL prefix of
http://localhost/scripts/runisa.dll?demo:RemotePreview::

If you were previewing the default.whteko file in the following folder layout:

```
c:\Clients                          ( umbrella folder for all clients and client's projects )
    \MyClient                       ( individual client folder )
        \Project1Name               ( project-folder = Dreamweaver site root )
            \Live                   ( folder to segregate files for production sever upload )
                \WebHub             ( parent folder for WebHub Application Server files )
                    \Apps           ( folder for WebHub Application EXEs )
                        Demo.exe
                    \Library        ( optional folder for DLLs and Delphi packages )
                    \whteko
                        \demo
                            WHAPPConfig_demo.xml
                            default.whteko
```

then, after copying the file to the Testing Server Folder (or Host Directory if using FTP), Dreamweaver would send the following URL to the browser:

http://localhost/scripts/runisa.dll?demo:RemotePreview::/Live/WebHub/whteko/demo/default.whteko

The WebHubApp with AppID 'demo', would run the code on the WebHub page with PageID of **RemotePreview**, which would read in the content of the file specified in the query string (after the pageID RemotePreview and two colons "::") (i.e. the App would read in the contents of **DWTestingServerURLPrefixMappedTo** + /Live/WebHub/whteko/demo/default.whteko). The contents of the Design Page specified in the default.whteko file would then be expanded and returned to the browser. (DWTestingServerURLPrefixMappedTo is a setting in the XML file of the WebHubApp and is set to match the Testing Server folder in Dreamweaver's Site definition.)

5. Save the Site Definition

To save the Site Definition including the Remote info and Testing Server configurations, click the [OK] button at the bottom of the Site Definition dialog.

Next up is exploring the rich feature set of the WebHub Designer.

4. Operation: Exploring Features

The *WebHub Designer* extension suite seamlessly integrates with Dreamweaver and provides you with access to all features via familiar mechanisms. The following discussion assumes Dreamweaver 8 is open and that the *WebHub Designer* extension has been installed.

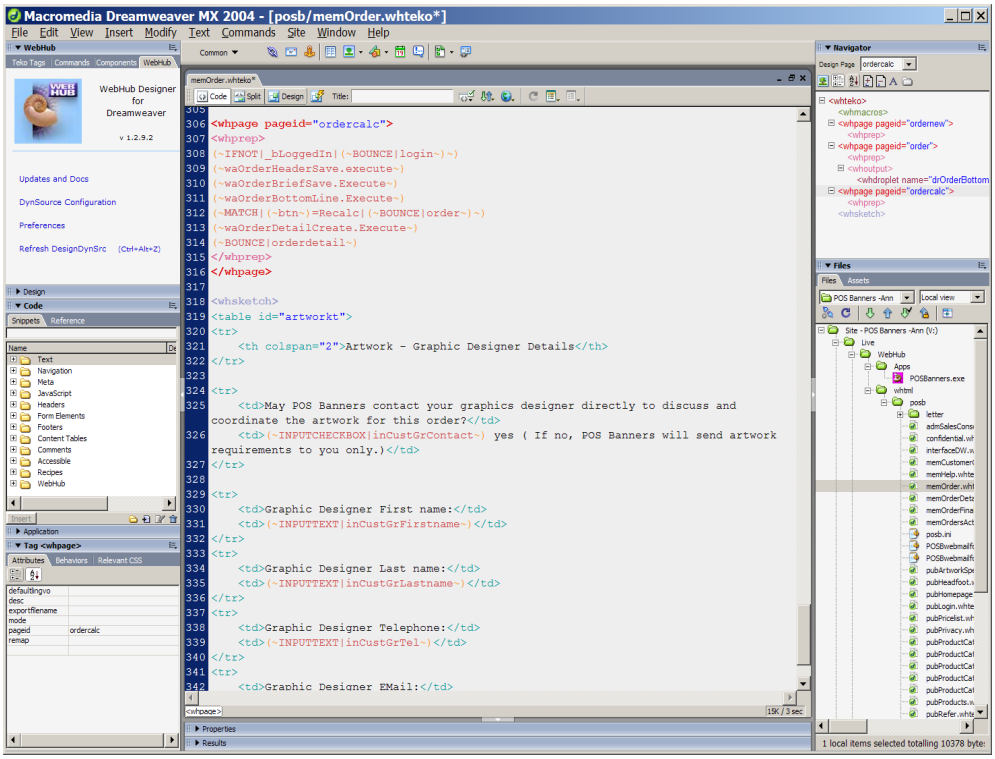
Docking the WebHub Designer Panels

Just after installation, your first step should be to **dock** the new floating panels. There is one panel-group named *WebHub* and one single-panel named *Navigator*. From Dreamweaver's main menu click **Window**, and you will notice 5 new menu items grouped together:

Figure 4-1 WebHub floating panel selections from the Window menu

Left-click **WebHub Main Panel** to make the *WebHub* panel-group visible, and left-click **WebHub Navigator** to make the *Navigator* panel visible. Using the ***gripper***  at the very top left of these floating panels, dock the *WebHub* and *Navigator* panels into an appropriate location within your left or right docked panel regions. See [Figure 4-2] for suggested panel positioning.

Figure 4-2 WebHub panel-group docked to the left, Navigator panel docked to the right



Accessing Features – a Summary

Access to functionality of the *WebHub Designer* is available through:

Floating panels:

WebHub panel-group

WebHub panel

Updates and Docs, DynSource Configuration, Preferences, Refresh DesignDynSrc Ctrl+Alt+Z

Teko Tags panel

listing and quick insertion of all **<whteko_tags>**.

Commands panel

listing and quick insertion of all built-in WebHub Commands.

Components panel

listing and quick insertion of the most commonly used WebHub components Properties and Methods.

Navigator panel

overview and easy traversal of code within a whteko file

Main Menu items:

File | **New** (General tab) Dynamic page WebHub v2 [Create]
File | **Open** Files of type:
All Documents (now includes *.whteko)
or WebHub v2 (*.whteko)
File | **Preview in Browser**
File | **Check Page** | **Validate as XML**
Edit | **Select Parent Tag**
View | **Design**
Insert | **Tag** WebHub tags
Modify | **Edit Tag**
Commands | **WebHub Designer** |
Refresh DesignDynSrc Ctrl+Alt+Z, **DynSrc Configuration**

Preferences, **Reload Configuration**
Site | **Manage Sites** [New] or [Edit] {Advanced tab} Testing Server
Window |
WebHub Main Panel, **WebHub Teko Tags**
WebHub Commands, **WebHub Components**
WebHub Navigator

Popup Menu items:
Edit Tag, Insert Tag, Edit WebHub Expression

Tag Inspector:
Attributes tab

Automatic features:
Code coloring of WebHub Stage 2 syntax
Expansion of WebHub Expressions into dynamic content in Design View
Assisted Tag attribute completion in Code view for all WebHub **<whteko_tags>**

Keyboard Shortcuts:

Ctrl+Alt+Z	Refresh the WebHub Application pointed to by the current designdynsrc
Ctrl+~	Toggle between Code-view and Design-view

Custom Panels

The WebHub Designer has 4 floating panels that have a very similar look and feel. They are named *Teko Tags*, *Commands*, *Components* and *Navigator.*

Each panel

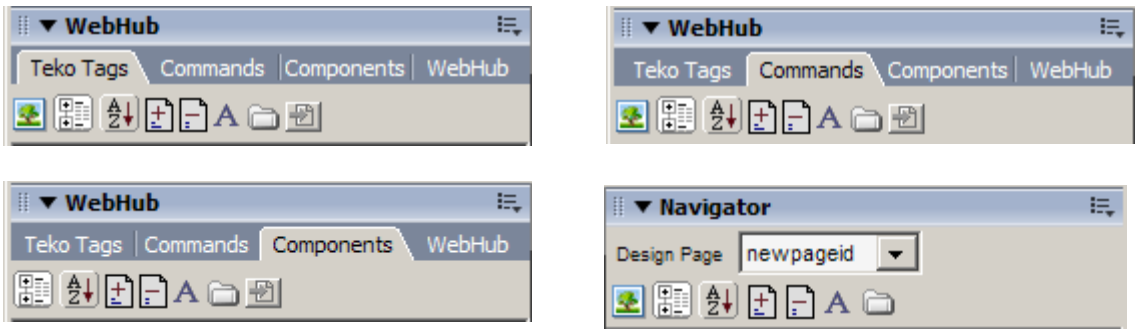
i) uses a similar set of custom toolbar icons.

ii) use a customized tree component for presenting content.

Custom-Panel Toolbar Icons

The toolbar portion of each of the WebHub Designer panels with tree contents use a similar set of custom toolbar icons as shown in [Figure 4-3]

Figure 4-3 Toolbar portion of Custom Panels in the WebHub Designer



Each toolbar icon is used in conjunction with the tree component that presents content in each panel. The function of each icon is as follows:

Toggle Tree Display

Clicking this icon toggles the display of content in the panel from a black and white tree presentation to a color tree presentation. The color tree presentation should be considered experimental, as it stretches the Dreamweaver Extension API beyond its intended capabilities, and it omits some features that are in the black and white version. See Custom Panel Tree Components below.

View by Category

Clicking this icon makes the panel content sort by Category

Currently Showing by Category

When the View-By-Category icon is showing with a white background then it is disabled and indicates that the panel content is already sorted by Category.

View Alphabetically

Clicking this icon makes the panel content sort Alphabetically. In the case of the Navigator Panel it makes the panel content sort “by Teko Tag” name.

Currently Showing Alphabetically

When the View-Alphabetically icon is showing with a white background then it is disabled and indicates that the panel content is already sorted alphabetically.

The icons for switching the sort sequence between Category and Alphabetical work as a pair, wherein at any one moment in time, if one is disabled thus indicating the current view, then the other is enabled.

So we have either  or 

Expand All Nodes

Clicking this icon fully expands all tree nodes for the displayed content.

Collapse All Nodes

Clicking this icon collapses tree nodes for the displayed content to a pre-determined minimum useful level. This does not fully collapse all nodes in the tree but instead collapses nodes to a level where it is considered that collapsing further would not represent a benefit.

Toggle Font Size

Clicking this icon repeatedly cycles through the font size choices for the displayed content.

Toggle Tree Icons

Clicking this icon toggles the display of folder and leaf icons for the content tree, on and off.

Insert Into Document - Active

Clicking this icon inserts the currently highlighted leaf node item (of displayed content tree) into the source code of the current document at the current cursor position.

Insert into Document - Disabled

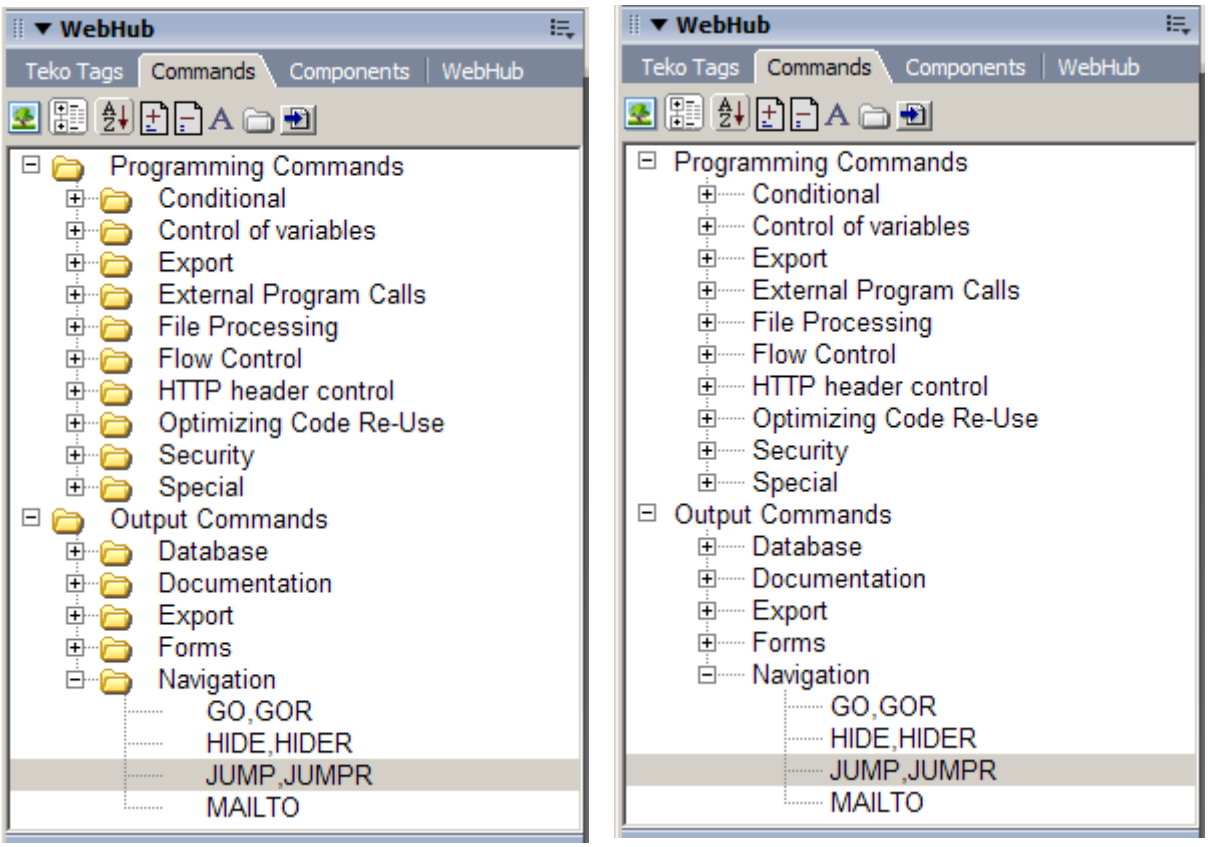
When the currently highlighted node is **not** a leaf node, the Insert icon is disabled.

Custom Panel Tree Components

Macromedia provides Dreamweaver Extension developers with a tree component for use in developing floating panel extensions. This component has been enhanced for use in the *WebHub Designer*, and in addition, a full color custom tree component has been created.

Although similar in content, the look and feel of the mono tree and the color tree are quite different.

Figure 4-4 The WebHub Commands Panel in mono, with and without tree node icons



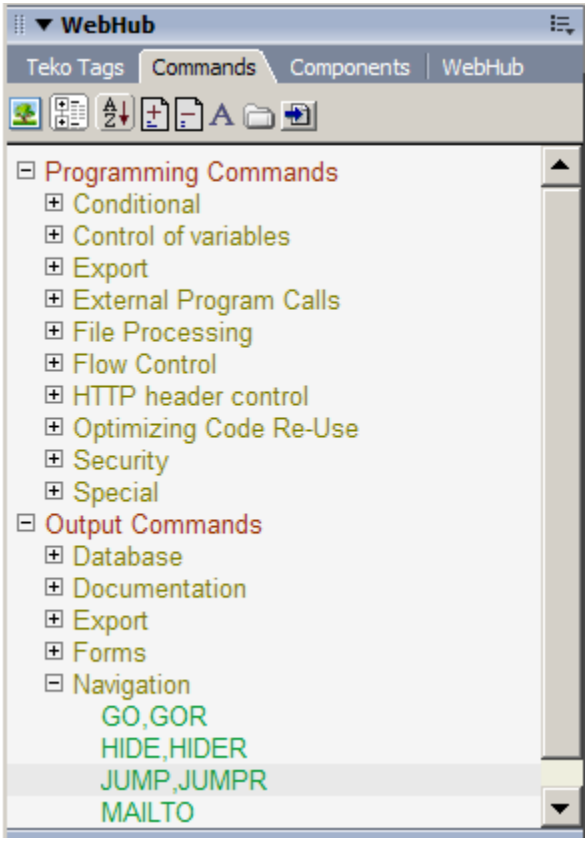
The mono version of the tree control is illustrated using the WebHub Commands panel as shown in [Figure 4-4].

Tree icons can be toggled on and off using the Toggle Tree Icons icon on the toolbar. Nodes of the mono tree can be double-clicked for insertion into the current document, or highlighted and inserted using the Insert icon at the right hand side of the toolbar icon group.

Once you have clicked into the tree content of a mono panel, the keyboard can be used to navigate and expand / collapse the tree. You can use the up, down, left and right arrow keys and the Enter Key. Pressing Enter on a branch node toggles its state between expanded and collapsed, and pressing Enter on a leaf node is the equivalent of clicking the Insert Icon. Note: sometimes you may need to click a tree node other than the one that is highlighted to get focus to transfer to a panel with a mono tree display.

Clicking the **Toggle Tree Display** icon on the toolbar will display the color tree component version of the same content. An example color display is shown in [Figure 4-5]

Figure 4-5 The WebHub Commands Panel in color



The color presentation of data is superior, but is should be considered as experimental because there is not currently sufficient functionality in Dreamweaver's Extension Interface to enable a fully functioning color tree to be designed.

The **minor** limitations of the color tree when compared to the mono tree are as follows:

- there is no keyboard navigation
- double-click on a leaf node does not work as a shortcut for clicking the Insert icon in the toolbar
- The thumb in the vertical scrollbar at the right on the panel is read-only i.e. it will correctly change size as expected when the content position and size changes, but it cannot be used to scroll the tree up and down. You can however use the top arrow and bottom arrow of the scrollbar as well as clicking the region of the scrollbar that is not occupied by the thumb.
- node icons are not used
- the + and – icons must be used to expand and collapse the branch nodes
- selection and highlighting of a node requires more precision as there is a small region just above and below each node that does not recognize the mouse click.

In all other respects the two tree components, mono and color, are more or less identical.

The exception is the Navigator panel which uses a more meaningful presentation in color.

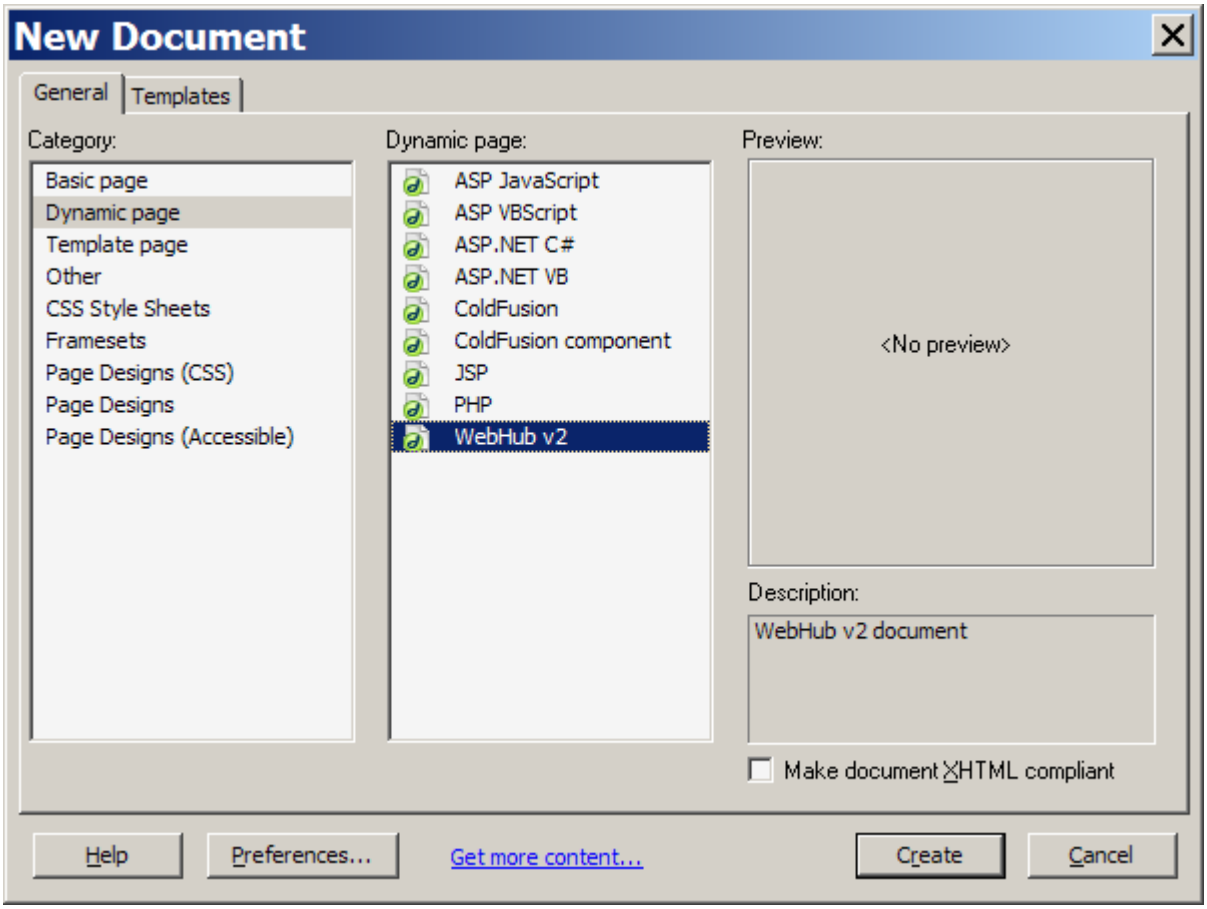
General Features

General features of the WebHub Designer are those that are not specific to Code view or Design view.

Creating a new whteko file

To create a new **`whteko`** file (a file containing WebHub pages), goto the Dreamweaver Main Menu and choose **File** | **New**. Then on the **General tab** select the *Category* of **Dynamic page** and then select the *Dynamic page* to be **WebHub v2**. Your *New Document* dialog should be identical to that shown in [Figure 4-6].

Figure 4-6 New Document dialog showing the selection of a WebHub 2 dynamic page



Ensure that the checkbox ☐ **Make Document XHTML compliant** is **NOT** checked, then click the [Create] button.

Dreamweaver will then create a new document in memory and label it Untitled-X (X=1,2,3...) until you save the document with a specific name.

The content of this new document is derived from a template file that was installed with the *WebHub Designer*.

The Code view of this new untitled document will be as follows:

```
<!DOCTYPE whteko PUBLIC "-//HREF//DTD whteko stage 2.14//Strict//EN//"
"http://webhub.com/dtd/0214/whteko.dtd">
<whteko
  designdynsrc="NotYetDefined"
  defaultlingvo="eng"
  designlingvo="eng"
  designpage="newpageid"
  designmode="code"
  showdoc="yes"
  showsketch="yes"
  notes="default WHTEKO file">

<whpage pageid="newpageid">

<whprep>          ← optional tag but recommended

</whprep>

<whoutput>        ← optional tag but recommended
<![CDATA[<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">]]>

<html>
<head>
  <title>Untitled Document</title>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"
/>

</head>
<body>


</body>
</html>
</whoutput>

</whpage>

</whteko>
```

Two aspects of this default document are discussed next: `designmode` and `<!DOCTYPE_tags>`.

About designmode

The whteko_tag attribute `designmode` has a value of `code`. In order to visually render code (that you type within the <body_tag>) using Dreamweaver's Design view, you will need to do the following:

i) Select the Dreamweaver Site that this document will become part of – Either choose **Sites | Manage Sites**, select your Site Name from the list and click [Done] or use the Files panel and select your Site Name in the drop-down above the panel's toolbar.

ii) change the value of the **<whteko_tag>** attribute `designdynsrc` from `NotYetDefined` to a valid *DynSource* for your Practice Server.

iii) change the value of the **<whteko_tag>** attribute `designmode` from `code` to `visual`.

About the <!DOCTYPE_tags>

The <!DOCTYPE_tags>, though not essential, are part of recommended best-practices. The new document template, by default, includes matching DOCTYPES that are compliant with XHTML 1.0 Strict.

The <!DOCTYPE_tag> at the **beginning** of the whteko file is unique to WebHub and is used for validation of contents for the entire whteko file. For validation to succeed, the **compliance of your html code must match** the XHTML standard referred to in the WebHub <!DOCTYPE_tag>. Also, if you use XHTML 1.0 Strict in the WebHub <!DOCTYPE_tag>, then you would use the XHTML 1.0 Strict <!DOCTYPE_tag> on your WebHub Pages.

To change to XHTML 1.0 Transitional instead of Strict, you would need to change both the WebHub <!DOCTYPE_tag> and the normal <!DOCTYPE_tags> on each page defined in the whteko file.

The WebHub <!DOCTYPE_tag> for XHTML 1.0 Transitional is

```
<!DOCTYPE whteko PUBLIC "-//HREF//DTD whteko stage
2.14//Transitional/EN//"
"http://webhub.com/dtd/0214/transitional/whteko.dtd">
```

and the html page <!DOCTYPE_tag> for XHTML 1.0 Transitional is

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

You will also notice that the <!DOCTYPE_tag> for the page is enclosed in a <![CDATA_tag>

```
<![CDATA[<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">]]>
```

This is to enable validation using the beginning WebHub <!DOCTYPE_tag> to work. When a WebHub Application expands the content for the WebHub page and sends the response to a browser, the surrounding <![CDATA[]]> tag is stripped away and only the contained <!DOCTYPE_tag> is sent with the html e.g. <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

Instead of the complexity of the <![CDATA[]]> tag on every page, we recommend that you define a macro for your chosen <!DOCTYPE_tag> in a shared whteko file, and then use a WebHub Expression to include it on all pages. e.g.

```
<whpage pageid="newpageid">
<whprep>
</whprep>
<whoutput>
(~mcDocType~)   ← the macro mcDocType is expanded
<html>
<head>
  <title>Untitled Document</title>
```

and in another whteko file loaded by the same application you would have:

```
<whmacros>
mcDocType=<![CDATA[<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">]]>
<whmacros>
```

Opening existing whteko files

To open an existing **whteko** file, you can

i) goto the Dreamweaver Main Menu and choose **File** | **Open**. Then on the Open Dialog that appears either choose *Files of type:* **All Documents** (now includes *.whteko) **or WebHub v2 (*.whteko)**. Select a displayed whteko filename and click [Open].

ii) goto the Dreamweaver Main Menu and choose **File** | **Open Recent** then **select** a filename form the recently opened list.

iii) Use Dreamweaver's File panel and open the whteko file from a folder within your Dreamweaver Site. After displaying the required filename (by expanding folders if necessary), either **double-click** the filename **or right-click** and **select Open**.

The recommended method (method iii) is to use Dreamweaver's File panel and open the document from within a Site, as it ensures that your document is opened with the **correct site selected** thus enabling visual design and previewing to work. The other two methods do not guarantee that you will also choose the correct Site for the file (either after or before opening it).

Refreshing a Design DynSource

Refreshing a Design DynSource (abbreviated to DesignDynSrc) means to **Refresh the WebHub Application EXE** that is pointed to by the *currently active design DynSource*.

The currently active design DynSource is the *DynSource* specified by the value of the **designdynsrc** attribute in the opening **<whteko_tag>** of the currently active document.

Refreshing a WebHub Application EXE means that you are requesting the WebHubApp EXE to **reload its XML file and ALL whteko files** that are listed in that XML file.

To refresh the *currently active design DynSource* you can: (all methods are equivalent)

i) click the hyperlink Refresh Design DynSrc Ctrl+Alt+Z on the *WebHub panel*

ii) goto the Dreamweaver Main Menu and choose **Commands** | **WebHub Designer** | **Refresh DesignDynSrc Ctrl+Alt+Z** or

iii) use the Keyboard Shortcut **Ctrl+Alt+Z**

Example: If the active Dreamweaver document was a whteko file and started with:

```
<!DOCTYPE whteko PUBLIC "-//HREF//DTD whteko stage 2.14//Strict//EN//"
"http://webhub.com/dtd/0214/whteko.dtd">
<whteko
  designdynsrc="demo-practice"
  designpage="newpageid">
```

then pressing **Ctrl+Alt+Z** would cause Dreamweaver to firstly examine the configuration of the DynSource named **demo-practice**. This configuration may be:

```
demo-practice=http://localhost/scripts/runisa.dll?demo|RemoteDesign|RemoteRefresh|1214|
(~
```

Dreamweaver would then refresh the WebHub Application EXE serving the AppID of 'demo', by sending an http request through the local WebServer (local because the domain in this DynSource configuration is localhost) to the WebHubApp. The URL of the request would be:

```
http://localhost/scripts/runisa.dll?demo:RemoteRefresh:1214
```

Preview in Browser

Dreamweaver's Preview in Browser feature is accessed from the Main Menu using
File | **Preview in Browser**

To preview a WebHub page using this feature requires

i) a correctly configured WebHub Application running in a Practice Server environment.

Correctly configured means that the WebHubApp has a *remote-preview* page defined and has an application default named **DWTestingServerURLPrefixMappedTo** set to the Practice Server Site Root (or a designated preview only folder) as well as correct security settings. See **Appendix E** – **Installing and configuring a WebHub Application EXE for use with Dreamweaver**

ii) a selected Dreamweaver Site with a correctly configured Testing Server. See **3. Installation and Configuration** – **Configuration** – **Creating and Configuring a Dreamweaver Site for WebHub**.

iii) that the page to preview is either the only page in the whteko file, or if more than one page exists then the page to preview is specified in the `designpage` attribute of the opening **`<whteko_tag>`**

Example: if two pages were defined within the one whteko file and they were named `default` and `contact`, and you wanted to preview the `contact` page, then the contact page would be specified in the opening **`<whteko_tag>`** as follows:

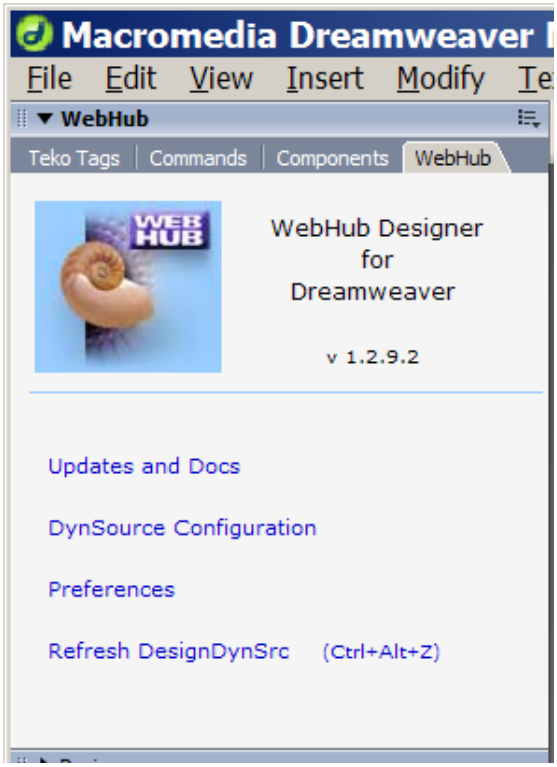
```
<whteko
    designdynsrc="demo-practice"
    designpage="contact"
.....>
```

This can be achieved either by editing the tag manually in Code view, or selecting the Design Page with the *Navigator* panel which automatically changes the `designpage` attribute value for you.

Updates and Docs

Updates and the latest documentation are available from the HREF Tools Corp. web site. To navigate to the correct page, use the Updates and Docs hyperlink on the WebHub panel.

Figure 4-7 WebHub Main panel



Once on the WebHub Designer electronic delivery page, you will be able to view or download:

WebHub Designer for Dreamweaver 8 User Guide (latest version of this guide)

WebHub Syntax Stage 2.14 Reference

Quick Reference for WebHub Commands

Code-View Features

Code view features, are those features of the WebHub Designer that are specific to Dreamweaver's Code view.

Syntax Highlighting

In addition to syntax highlighting for normal html, javascript and stylesheet code, the WebHub Designer also adds color syntax highlighting for all

- **<whteko_tags>**
- WebHub Expressions e.g.

  (~JUMP|contact|Contact Us~)
- Parentils (~ ~) and
- WebHub hidden comments

  <!--- -->

A sample of most highlighting enhancements is shown in [Figure 4-8] to the right.

Figure 4-8 Syntax Highlighting

Navigation

The *Navigator* panel, when used with Dreamweaver's Code view, enables you to:

1. **see a complete overview of <whteko_tags> in the current document**

   (this can be as displayed as an outline view or sorted by main whteko tag type)

2. **easily traverse code within a whteko file**

   (just click on a **<whteko_tag>** in the navigator and the cursor is moved to that tag's position in code view)

3. **select the current Design Page**

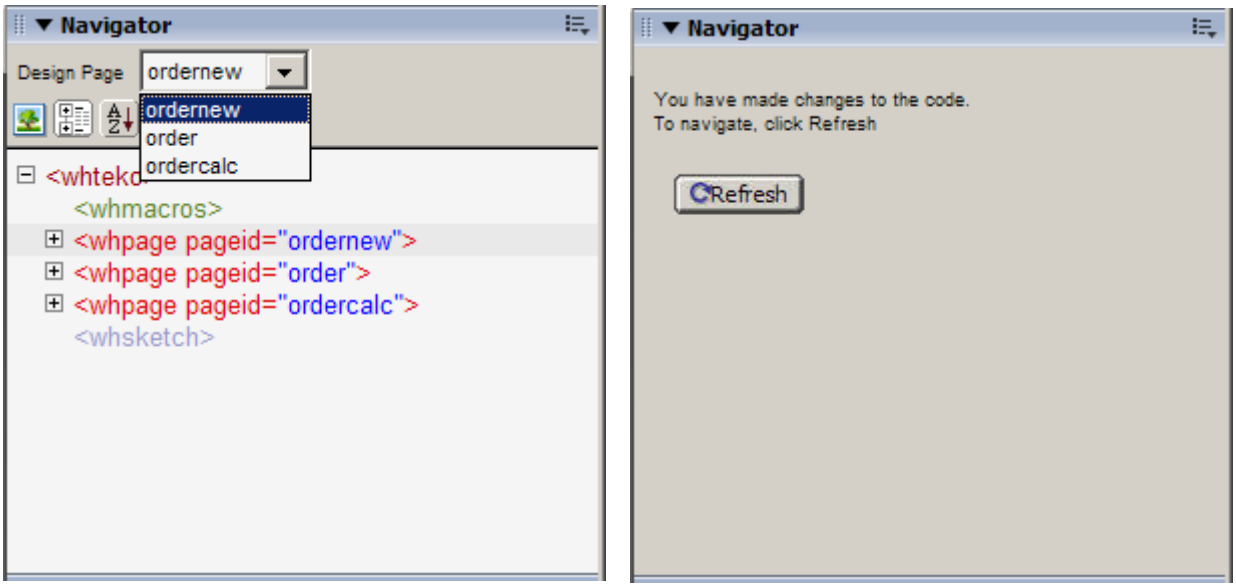
   (This changes the designpage attribute in the opening **<whteko_tag>** to the selected PageID in the Design Page drop-down. In code view this is mostly used to select a current page for previewing)

Importantly, please note that the *Navigator* always maintains the correct value of the designpage attribute in the opening **<whteko_tag>**. If you **change** the **Design Page** using the drop-down at the top of the Navigator, then because the designpage attribute also changes, the code of the current document will be in an **unsaved state**. Furthermore, if you open the document with an invalid designpage attribute, the Navigator will change the attribute, as the document is opened, to be the value of the PageID in the first **<whpage_tag>** declared in the file!!

As you move the cursor position in Code view, the Navigator tracks this position and automatically highlights the current **<whteko_tag>** that the cursor is within.

If changes are made to the source code in Code view, then the Navigator is grayed out and displays a refresh button. This is for performance reasons and is similar in approach to other built-in Dreamweaver panels.

Figure 4-9 Changing the Design Page in the Navigator | Refresh view when changes are made



The following figures illustrate the various views that the *Navigator* panel can display for a sample whteko file with 3 pages, a droplet and a macro definition.

Firstly the Outline view.

Figure 4-10 Navigator panel with sample outline (collapsed) view – color and mono

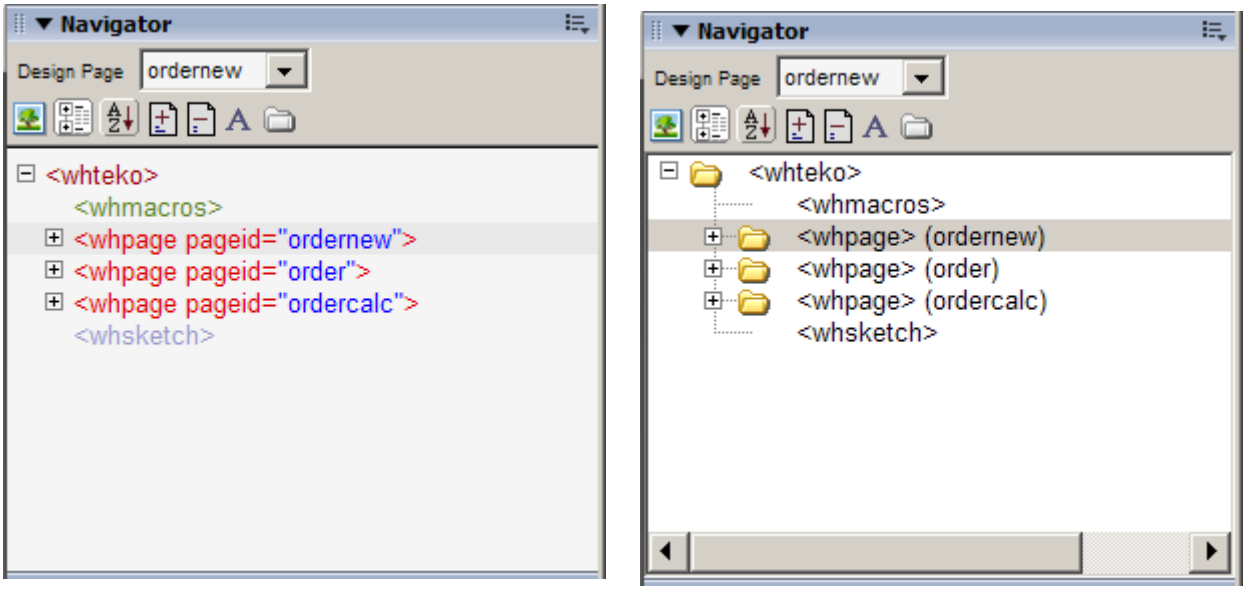
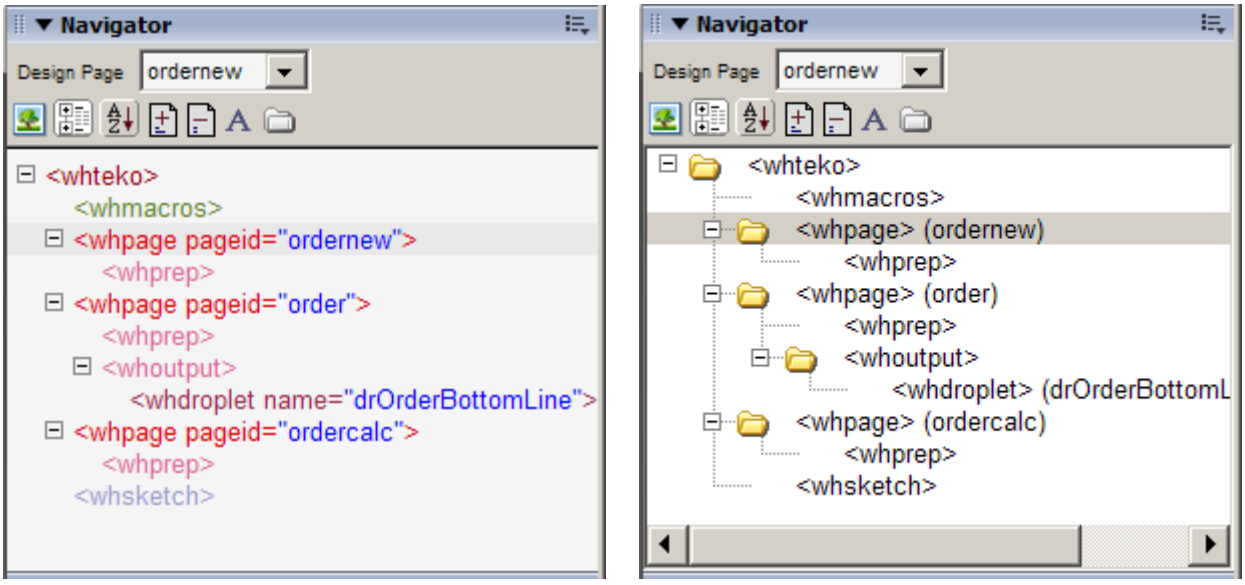
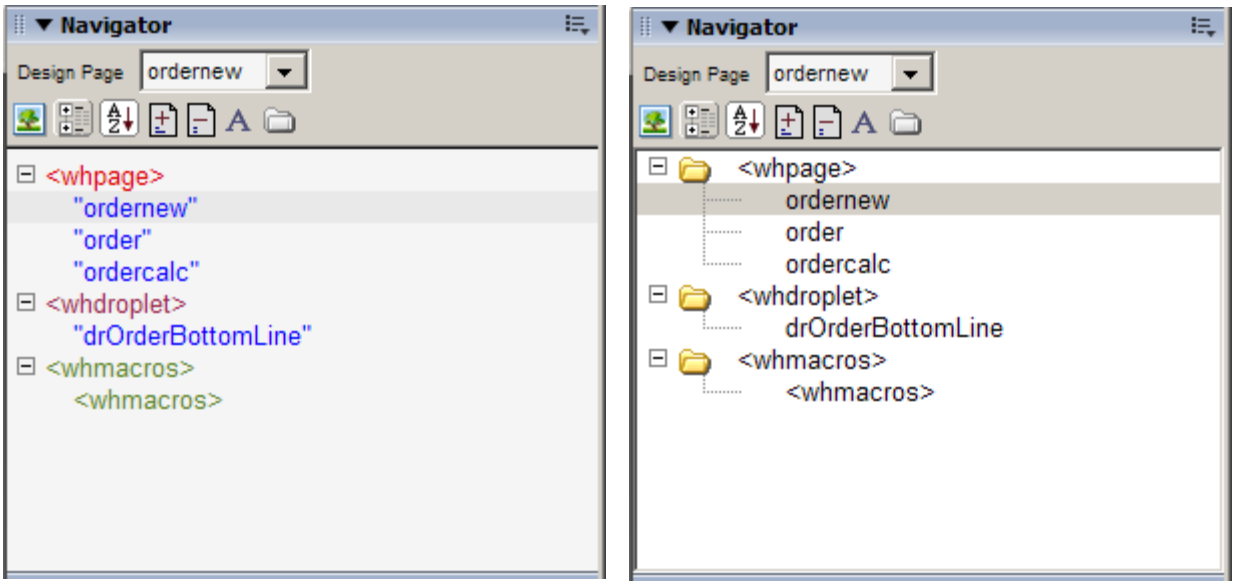


Figure 4-11 Navigator panel with sample outline (expanded) view – color and mono



Secondly the Tekero tag view. This view groups together by tag type, all tags that can be direct children of the main **<whteko_tag>**.

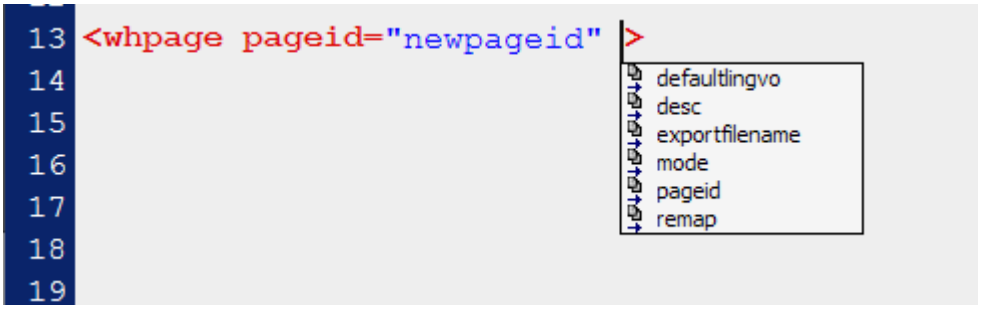
Figure 4-12 Navigator panel with sample Tekero view – color and mono



Code Hints for <whteko_tags>

Dreamweaver's built-in ability to provide code hints is also available for all **<whteko_tags>**. The code hints feature helps you insert and edit code quickly and without mistakes. When you type (in Code view) the first letters of a **<whteko_tag>**, or while within a **<whteko_tag>** you type the first letters of an `attribute name`, a list appears, suggesting options to complete your entry. You can use this feature to insert or edit code, or just to see the available `attributes` for a **<whteko_tag>**.

Figure 4-13 Code Hint for the attributes of a <whpage_tag>



WebHub Teko-Tag insertion and editing

A WebHub Teko Tag is any tag beginning with **<wh**.

The set of Teko tags for WebHub Syntax Stage 2.14 are presented in order of priority, rather than alphabetically, in the following table.

Teko tag	*usage*
<whteko>	always required - top level xml tag
<whpagesettingslist>	optional - used for shared page settings
<whpage>	required for page declarations
<whpagesettings>	optional – used for custom page settings
<whdroplet>	required for page part declarations
<whmacros>	required for macro definitions
<whrow>	optional – used with TWebActions that output HTML <table...> tags
<whtranslation>	optional - used with the new language translation feature
<whtranslations>	optional - used with the new language translation feature
<whprep>	optional - for organizing W-HTML
<whoutput>	optional - for organizing W-HTML
<whdesign>	optional - used during design only
<whdoc>	optional - used for documentation
<whsketch>	optional – used for html sketches
<whremote>	optional – used for secondary source dynamic content
<!--- -->	optional - used for hidden comments
[~ ... ~]	optional - used with the new language translation feature

Inserting a Teko Tag

To insert a Teko tag with assistance, firstly position the cursor in Code view where you want to insert the tag, then use one of the following equivalent methods to insert the tag or to launch a Tag Editor dialog:

1. Launch the Tag Chooser by either 1. going to Dreamweaver's Main Menu and choosing **Insert** | **Tag** or 2. right-clicking in Code-view and choosing **Insert Tag** form the Pop-up Menu. Then on the Tag Chooser dialog underneath *Markup Language Tags* in the tree-outline in the left pane, choose *WebHub Tags*, then double-click the name of the tag (listed in the right-hand pane) that you want to insert.

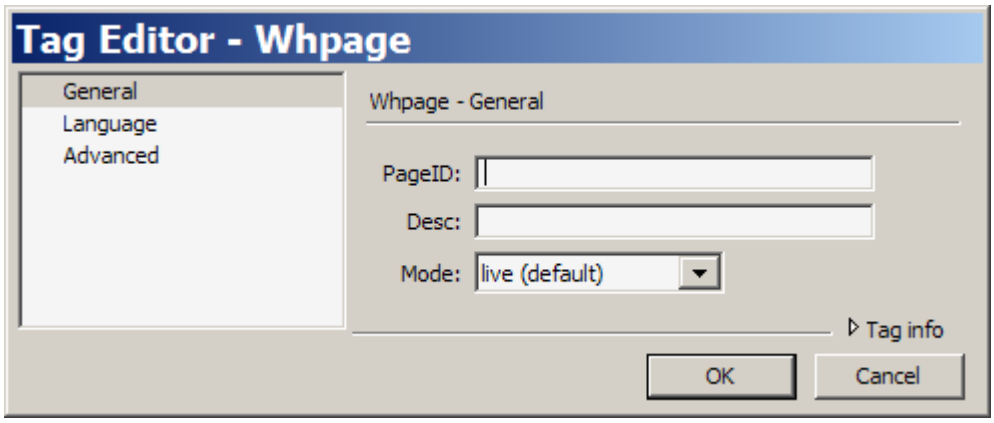
   OR

2. Use the *Teko Tags* panel. If you are using the mono version of the panel, then double-click the required tag name. If using the color version, first click the tag name then click the Insert toolbar icon at the top of the Teko Tag panel

If the Teko tag has no attributes, then it will be inserted immediately (both an opening and closing tag will appear in Code view)

It the Teko tag has possible attributes, then a Tag Editor dialog will appear and this dialog will be specific to the selected tag.

Figure 4-14 Tag Editor dialog for the Teko tag <whpage>



Editing a Teko Tag

To edit a Teko tag that has possible attributes, other than manually, use the Tag Editor dialog.

Launch this dialog by positioning the cursor anywhere within the opening tag of the Teko tag you wish to edit, then either 1. goto Dreamweaver's Main Menu and choose **Modify** | **Edit Tag** or 2. right-click in Code-view and choose **Edit Tag** from the Pop-up Menu.

The same Tag Editor dialog that appeared during tag insertion is used, except this time any attributes values that existed in the tag in Code view are also displayed in the dialog.

Note: The Tag Inspector panel is sensitive to locked-regions used by the WebHub Designer and is not always available to edit Teko tags.

WebHub Command insertion and editing

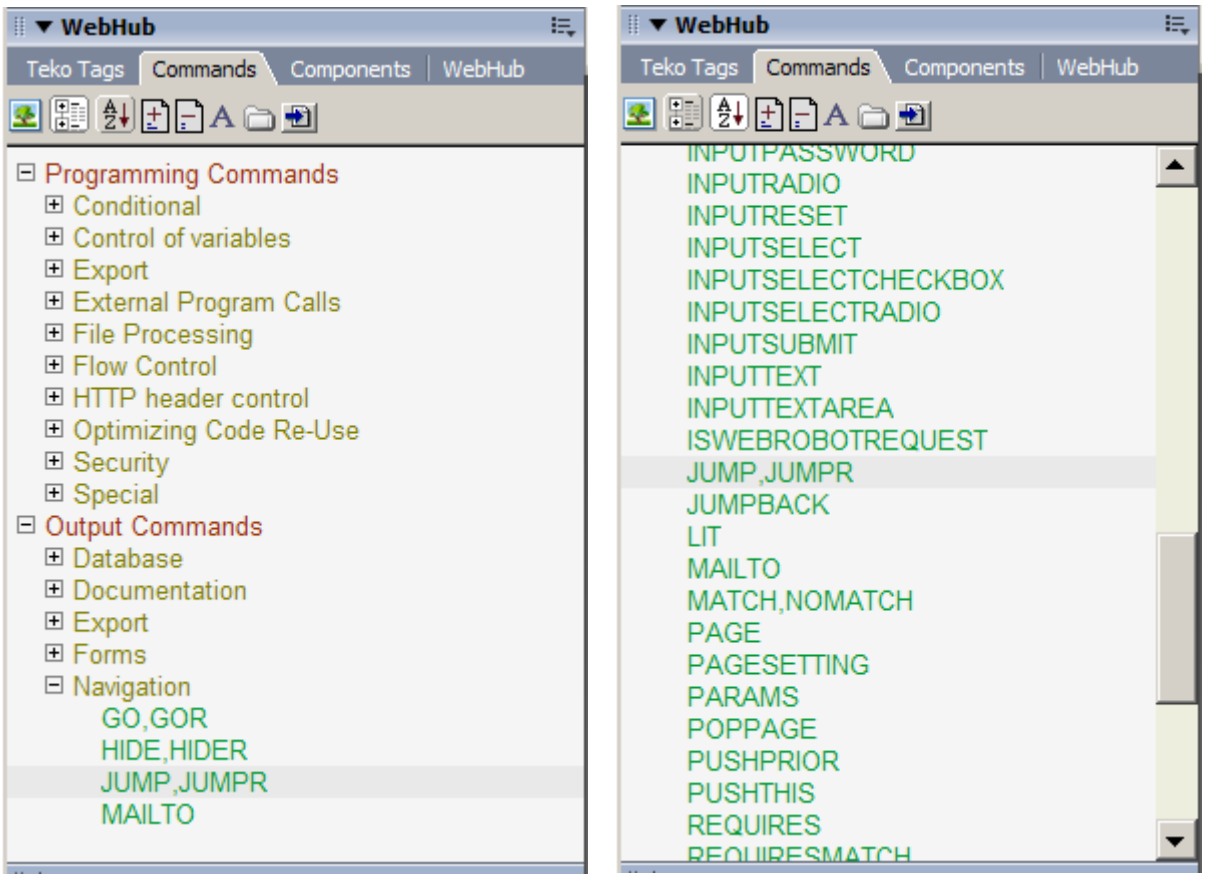
Inserting and editing WebHub Commands can either be approached manually or by using the WebHub Expression Editor. Due to their complexity, WebHub Commands do not have the standard code hints.

Inserting a WebHub Command

To insert a WebHub Command with assistance:

1. position the cursor in Code view where you want to insert the WebHub Command
2. Display the *Commands* panel (if it is not already showing) and use this to locate the particular WebHub Command you wish to insert. There are two sorting methods available in the Commands panel, one is a Category view and the other is an Alphabetically sorted listing of the built-in WebHub Commands. [Figure 4-15] shows the WebHub Command JUMP being located using each sort method on the color version of the panel.

Figure 4-15 Locating the WebHub Command JUMP using the Commands panel (both views)

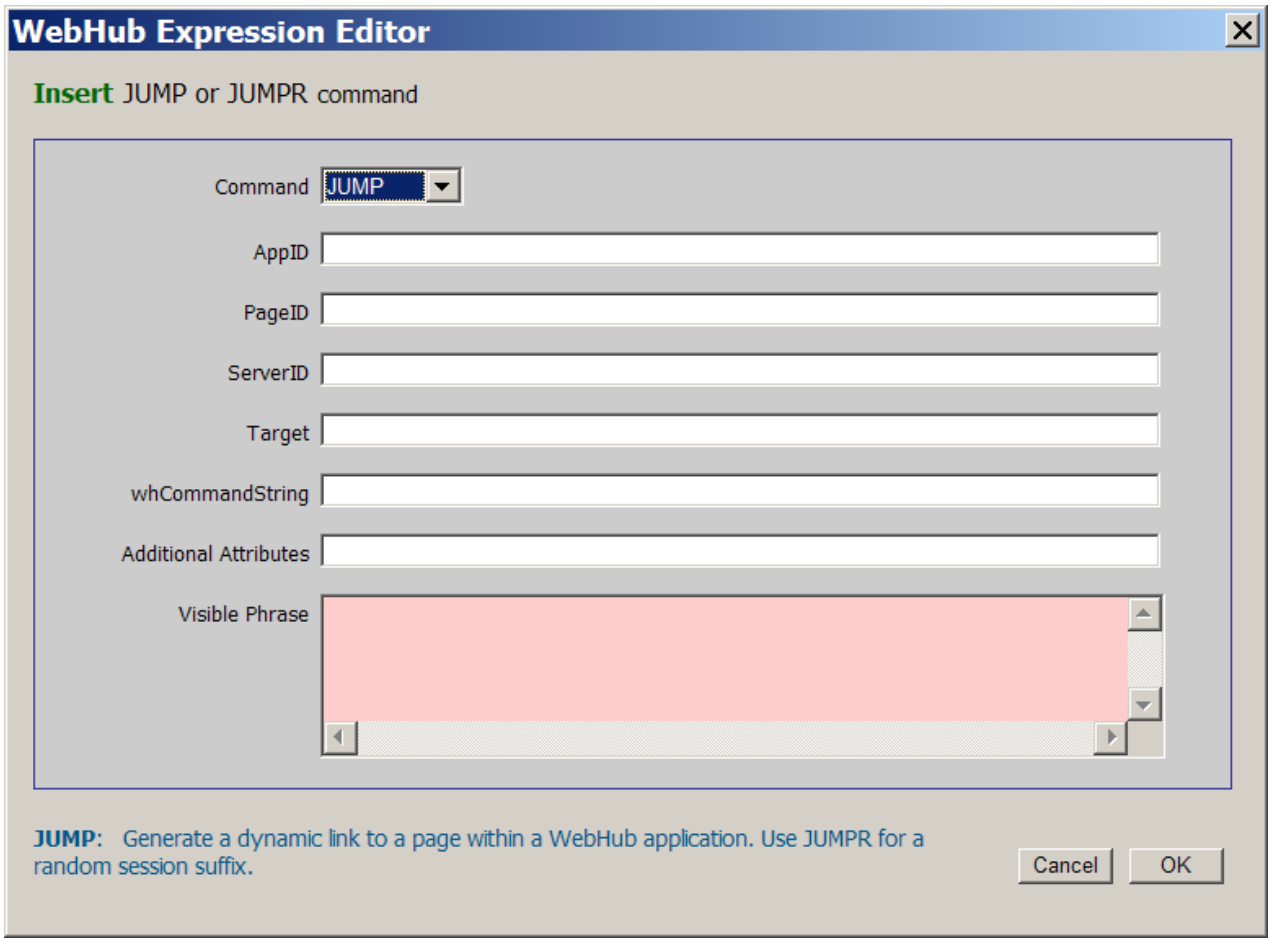


.

3. If using the **color** version of the *Commands* panel, click the required Command then click the **Insert toolbar-icon** (located at the right hand side of the toolbar at the top of the panel). If using the mono version of the panel, just double-click the required command.
4. If the Command has no optional parameters, then it will be inserted immediately, otherwise the WebHub Expression Editor will be displayed.

The WebHub Expression Editor is fully customized for each WebHub Command that can have parameters. Figure 4-16 shows the Editor when initially inserting a JUMP command.

Figure 4-16 Inserting a JUMP command with the WebHub Expression Editor



5. Fill in parameter values for the Command you are inserting. Parameters with a light pink background are **required** parameters, all others are optional.
6. Click the **OK Button** on the WebHub Expression Editor dialog to insert the WebHub Command (complete with the parameters that you entered) into your document's code..

Example: Assume that you entered the following values into the WebHub Expression Editor dialog when inserting a JUMP command:

AppID:	demo
PageID:	contact
Visible Phrase:	Contact Us

Clicking the OK Button on the editor dialog would insert the following into your document code:

```
(~JUMP|demo:contact|Contact Us~)
```

Editing a WebHub Command

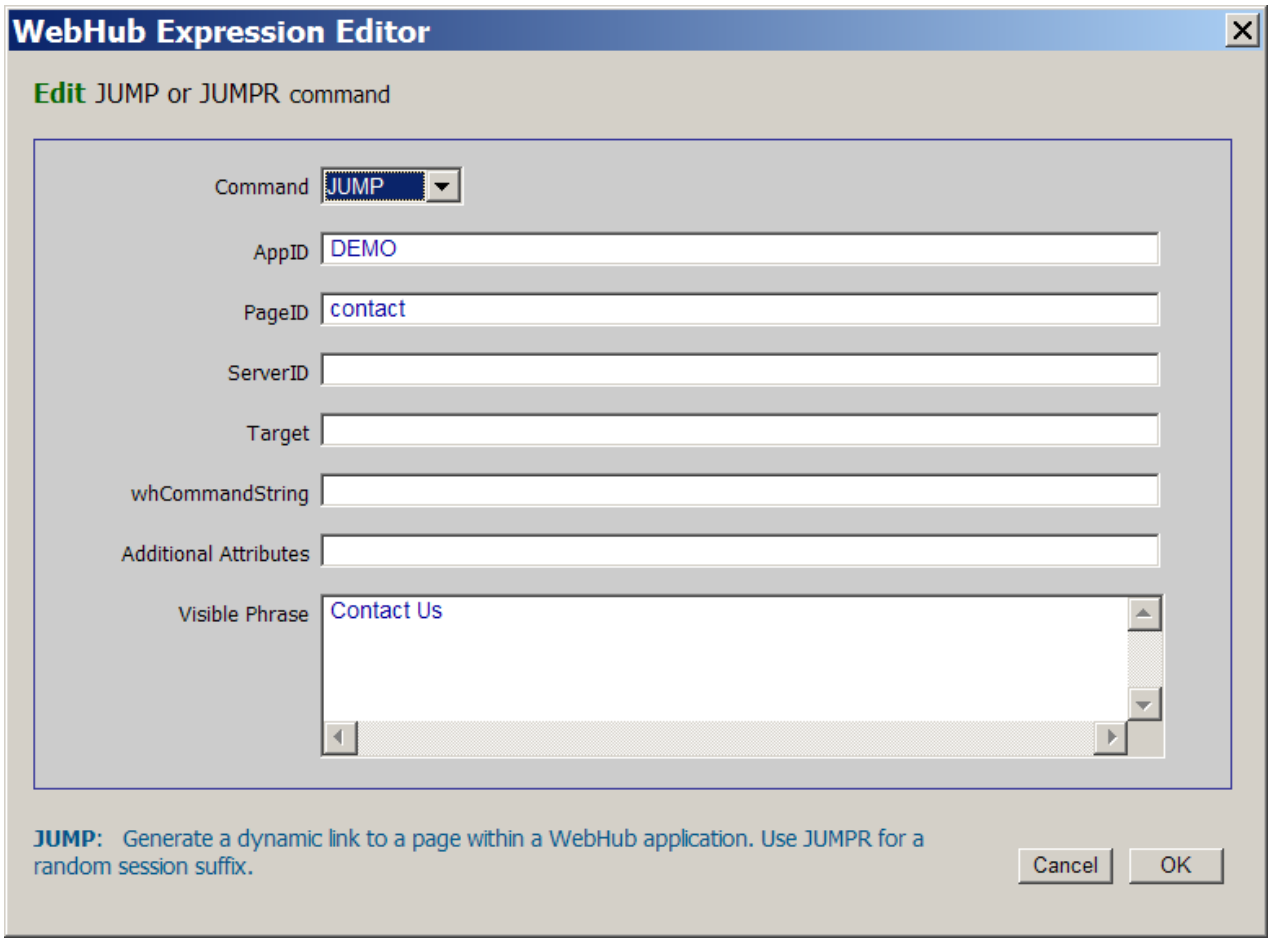
To edit a WebHub Command with assistance (i.e. using the WebHub Expression Editor rather than editing the code manually):

1. position the cursor (in Code view) on the Command name at the beginning of the expression that you want to edit. Commands nested to any level within the Expression can be edited.
2. right-click to display the Pop-up menu, then choose the menu-item **Edit WebHub Expression**. The same WebHub Expression Editor that is used to insert a Command is displayed, but this time existing parameter values for the chosen command appear in the dialog next to their respective labels.

As an example consider the expression `(~JUMP|demo:contact|Contact Us~)`

Right clicking on the word `JUMP` in the expression and choosing **Edit WebHub Expression** would display the dialog shown in [Figure 4-17].

Figure 4-17 Example of editing a JUMP command with the WebHub Expression Editor



WebHub Components - Properties and Methods

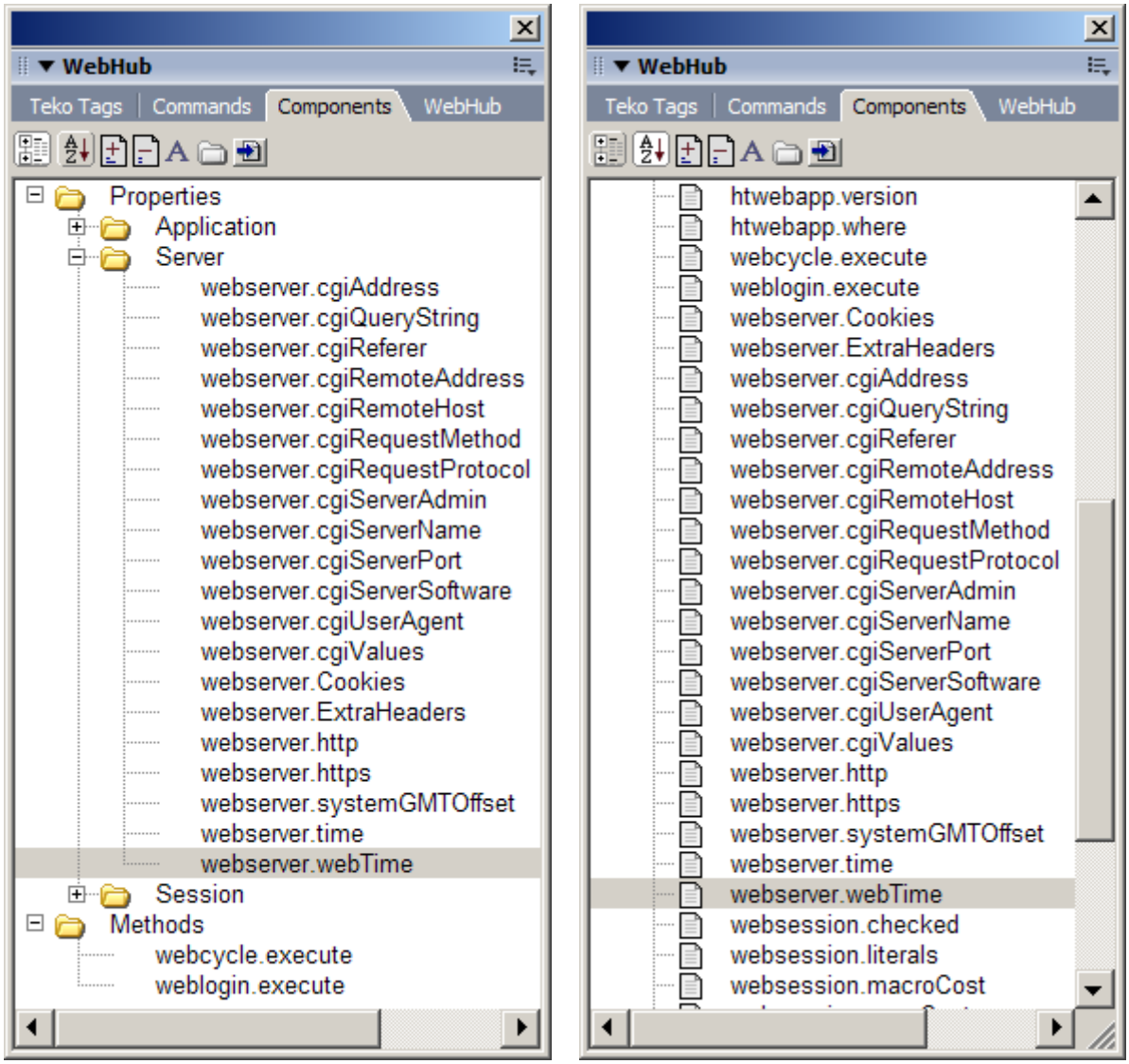
Inserting WebHub Components properties and methods can either be approached manually or by using the *Components* panel of the WebHub panel group. Editing is always done manually.

Inserting a WebHub Component Property or Method

To insert a **commonly used** WebHub Component Property or Method with assistance:

1. position the cursor in Code view where you want to insert the WebHub Component Property or Method.
2. Display the *Components* panel (if it is not already showing) and use this to locate the particular commonly used WebHub Component Property or Method you wish to insert. There are two sorting methods available in the Components panel, one is a Category view and the other is an Alphabetically sorted listing of common WebHub Component Properties and Methods. [Figure 4-18] shows the property **WebTimeGMT** for the WebHub Component **CentralInfo** being located using each sort method on the mono version of the panel.

Figure 4-18 Locating property CentralInfo.WebTimeGMT using the Components panel (both views)
NB: In earlier versions of WebHub, CentralInfo.WebTimeGMT was named webserver.webTime



The WebHub Expression Editor

The WebHub Expression Editor is a dialog used to insert and edit WebHub Commands. These Commands can be stand-alone or nested within WebHub Expressions.

WebHub Expressions Explained

WebHub Expressions are used within HTML markup:

1. to dynamically insert content into a web page while it is being generated by a WebHub Application (WebHubApp), and
2. to interact directly with a WebHubApp's logic and data.

A WebHub Expression is any sequence of commands and nested expressions surrounded by parentils.

Parentils are the delimiters (~ and ~) . These are used to distinguish a dynamic WebHub expression from the surrounding html and are used for nesting expressions within expressions.

There are 5 basic types of atomic (non-nested) WebHub Expressions.

Command	a built-in or a custom command	e.g. (~CLEAR|~)
Tekero	a page, droplet, translation or macro	e.g. (~drPageHeader~)
Session variable	a SessionLiteral or SessionBoolean	e.g. (~UserName~)
WebAction Method	a method of a WebAction component	e.g. (~waWebAction.Execute|~)
Component Property	a property of a Delphi component	e.g. (~CentralInfo.WebTimeGMT~)

Expressions can be nested to a depth of 40.

Editable Expression Types

The WebHub Expression Editor dialog is used at present only to insert and edit WebHub Commands.

Tekeros, Session variables and Component properties do not have any parameters to edit. WebAction Methods on the other hand generally have parameters but are not editable at present with the WebHub Expression Editor. A future version of the WebHub Designer will address this.

WebHub Expression Editor Access

The WebHub Expression Editor is automatically displayed when inserting commands from the WebHub Command floating panel, and is accessed for editing by placing the Dreamweaver source code cursor on the command to be edited, right-clicking and selecting **Edit WebHub Expression** from the popup menu.

Note: Commands embedded within other commands can be edited separately if required.

WebHub Expression Editor Features

The WebHub Expression Editor is fully customized for each built-in WebHub Command that can have parameters.

Features include:

1. Syntax validation of command parameters.

2. Color highlighting of required parameters (with a light pink color)

3. Parameter purpose hints (shown at the bottom of the dialog) describing the purpose of each parameter.

4. Parameter expansion hints (shown at the bottom of the dialog) describing the expansion method used by WebHub for each parameter. This is displayed when the cursor is placed into the edit box for a given parameter.

   The expansion hint is one of the following:

   - This parameter is always expanded
   - This parameter is explicitly expanded (only if surrounding parentils are used)
   - This parameter is never expanded.
   - This parameter is conditionally expanded (see help for this command)

5. Conversion of cryptic enumerated choice characters into simple-to-read drop-downs.

   For example in the MATCH command, the comparison operator is translated as follows:

Character	**Drop-down phrase in the WebHub Expression Editor**
=	case sensitive match
~	case insensitive match
!	starts with
[	is contained in
$	is distinctly in

Code Validation

By specifying one of two WebHub specific <!DOCTYPE tags at the start of a whteko file, it is possible to use Dreamweaver's built in **Validate as XML** feature to validate the syntax of the whteko file contents, including all contained xhtml page output code.

See **6. Application Programming: – Designing Dynamic WebHub Page Content – The DOCTYPE tag and validation** for a more complete discussion of the choice and use of the DOCTYPE tags and their use with validation.

Access to code validation is via Dreamweaver's main menu:

File | **Check Page** | **Validate as XML**

Advanced use of WebHub expressions within the attributes of html tags may require enclosure within a CDATA tag i.e. `<![CDATA[ ]]>`, in order for the validation test to pass successfully.

Design-View Features

Design view features, are those features of the WebHub Designer that are specific to Dreamweaver's Design-view.

The WebHub Designer enables visual design of page layout and contents for dynamically generated WebHub pages. For the selected **design-page** in a whteko file, this is achieved through real-time, unaltered, complete code rendering of dynamic-content and is displayed in Dreamweaver's Design-view.

Visual Design Prerequisites

Enabling the visual design of a WebHub page when using Dreamweaver requires the following prerequisites:

1. a correctly configured Dreamweaver Site (for access to static resources)
2. a running WebHub Application EXE
3. a *DynSource* configured to access this WebHubApp
4. a `designdynsrc="DynSourceName"` attribute in the **`<whteko_tag>`** at the start of the document, where `DynSourceName` is the alias for the *DynSource* in 3.
5. a `designpage="PageID"` attribute in the **`<whteko_tag>`** at the start of the document where `PageID` is the `pageid` attribute value of the **`whpage`** you are designing.
6. that the **`<whteko_tag>`** have have a `designmode` attribute of `designmode="visual"`
7. a Dreamweaver specific setting of all required session variables that may be required for the page to display correctly. The can be achieved using a combination of the IFINDW command and the <whdesign_tag>.
8. Optional Dreamweaver specific <whdesign_tag> content that only displays in the visual design mode of Dreamweaver

When all of the above are in place, then switching to Design View in Dreamweaver will trigger the WebHub Designer to display dynamic content.

Note: Sometimes a change may need to be made to a document's code (any change at all) in order to trigger Dreamweaver to run the translator feature of the WebHub Designer. This is a Dreamweaver API limitation rather that a deficiency of the WebHub Designer.

Selecting a Design Page

When more than one page is defined in a whteko file, then a page must be selected to visually design (or view) in Dreamweaver's visual design mode.

There are 3 ways to select a page.

1. By default, if no page is specified in the `designpage` attribute of the **`<whteko_tag>`**, then the first page in the whteko file is selected.
2. manually change the `designpage="PageID"` attribute in the **`<whteko_tag>`** at the start of the document to be the `pageid` attribute value of the **`whpage`** you are designing.
3. Use the Design Page selection drop-down at the top of the Navigator panel.

Viewing Dynamic Content

With visual design prerequisites in place, viewing dynamic content for the selected design page is achieved by triggering Dreamweaver's so-called "translator" feature. (This feature has nothing to do with multilingual translations; it is a "translation" from code-view to design-view.) For whteko documents, this is a call by Dreamweaver to a javascript function defined in the WebHub Designer.

The translator is triggered by particular events within Dreamweaver, and is always associated with synchronizing code view and design view.

So when you are designing, switching to Design-view from Code-view is the only way to trigger the translator. The keyboard shortcut for this is **Ctrl+~**

Importantly, when changes are made to the WebHub Application or droplets or content in a whteko file **other** than the one you are designing, the only way to see that change in Dreamweaver's visual mode is to make a change to the source code in the file you are currently working with. This is required to trigger the translator to show the externally modified new dynamic content.

Editing Content in Design-view

Assuming `designmode="visual",` when Dreamweaver (via the WebHub Designer translator) presents content in Design View, there are two types of content to consider.

The first type of content is fully editable html tags and text. Editing is as per normal.

The second type of content is read-only. These are regions of the Code-view source that are locked by Dreamweaver and are not able to be edited. These regions can be highlighted and deleted, but not changed. Be careful of accidental deletion which can be caused by hurrying through mouse-clicks and keystrokes when editing in design mode.

All content that is dynamic (as defined by WebHub expressions in code View) is locked and read-only.

Navigation in Design-View

When navigating in Design view, select the current design page by using the Design Page drop-down at the top of the navigator, **not** by double-clicking a different **`<whpage_tag>`**.

Clicking on tags in the navigator that are outside the currently displayed design page will switch the view back to Code-view and position the cursor just as in Code-view navigation.

Rapid visual design of multilingual content

WebHub Syntax Stage 2 is designed to facilitate rapid development of multilingual content.

The designlingvo attribute

In Dreamweaver, to view previously prepared multilingual content in Design View, simply set the `designlingvo` attribute in the opening **`<whteko_tag>`**.

For example, the following whteko_tag settings would cause all dynamic language translation elements to display in English when switching to Design View:

```
<whteko
  designdynsrc="demo-practice"
   defaultlingvo="eng"
   designlingvo="eng"
   designpage="Customer"
   designmode="visual">
```

To have all dynamic language translation elements display in French, just change the `designlingvo` attribute to

```
designlingvo="fra"
```

then switch back to Design-view.

The defaultlingvo attribute

The other language attribute `defaultlingvo` is used to set a default for language translation declarations within the whteko file. This means that if a specific language is not specified in the `lingvo` attribute of the WebHub language translation elements, then the defaultlingvo setting is used by default.

WebHub language translation elements

There are 3 language translation elements

```
<whtranslation key="" lingvo="" show="">
Text
</whtranslation>

<whtranslations lingvo="">
~key=text pairs
</whtranslations>
```

`[~key~lingvo~`Text to display**`~]`** - this is a shortcut for the **`<whtranslation_tag>`**

The syntax and usage of these elements is covered in detail in the document describing

WebHub Syntax Stage 2. See **7. Support** – **Resources**.

DW features to avoid when editing whteko files

There are a few built-in Dreamweaver features that are not compatible with whteko file syntax. This is because html/xhtml content can occur in multiple pages within whteko syntax, whereas Dreamweaver normally assumes a single page design per file.

Here is a list of the main features to avoid:

Working with the XHTML standard

1. When selecting

   File | **New** {General tab} *Category*: Dynamic Page *Dynamic Page*: WebHub v2

   make sure you **do NOT check** ☐ *Make document XHTML compliant*

   (This is because checking this option causes Dreamweaver to replace the whtml <!DOCTYPE tag at the start of the whteko document, with an XHTML html <!DOCTYLE tag. Code validation will not be possible unless the whtml <!DOCTYLE tag is used)

2. Do not use **File** | **Convert** | **XHTML**

Validating Markup

Do not use **File** | **Check Page** | **Validate Markup**

Instead use **File** | **Check Page** | **Validate as XML**

Viewing Live Data

The following are not used with WebHub pages:

View | **Live Data**
View | **Live Data Settings**

The need for Live Data is obviated by the WebHub Designer's use of dynamic content in Design-view. Live Data is unnecessary and not used by the WebHub v2 server model.

Use the **<whteko_tag>** attribute `designmode="visual"` and switching from Code view to Design view to see live data for a WebHub page.

Embedded style elements

Embedded style elements (<style type="text/css">...</style>) in the <head_tag> area of a page are ok but **do not use WebHub Expressions within the style element unless they are in a droplet**.. If expressions are required, then create a separate droplet to contain the style content, then call the droplet from within the <head_tag>.

e.g.

```
<whdroplet name="drStylesDefault">
<style type="text/css">
#content {
        background-color: (~mcColDefaultBg~);
</style>
</whdroplet>

<whpage pageid="default">
<whoutput>
(~mcDocType~)
<html>
<head>
  <title>Document title</title>
      (~drStylesDefault~)
and so on
```

This technique will prevent a Dreamweaver translation idiosyncrasy from polluting your source code with strange lock code tag syntax.

Automatically updating links

Whenever Dreamweaver asks if you would like to update links, say after moving files within a site, **always say No**. WebHub links are mostly dynamic and should not be automatically massaged by Dreamweaver.

5. Customization

Preferences

The *WebHub Designer* can be customized for general look and feel preferences..

Access

Access to the *WebHub Designer Preferences* dialog is via:

WebHub panel-group: *WebHub panel* click Preferences
or Main Menu item: **Commands** | **WebHub Designer** | **Preferences**

This will launch the dialog named *WebHub Designer Preferences* [Figure 5-1].

Figure 5-1 WebHub Designer Preferences dialog

Settings

The preference settings currently available are font sizes for the floating panels and dialogs used by the *WebHub Designer*, and also whether a confirmation message is required after successfully refreshing a WebHub Application (*DynSource*).

The choices for each setting are:

Dialog Font Size: *smallest, smaller, normal, large, larger, largest*

Default font size for the WebHub Expression editor dialog.

Panel Font Size: *normal, large, larger, largest*

Default font size for WebHub Designer floating panels.

Confirm Refresh: *true, false*

If true, show a message upon successful refresh of a DynSource.

Pressing the [Save] button writes your preferences to the WebHub Designer configuration file.

6. Application Programming

Review of Basics

Before considering the visual design of WebHub pages, it is worthwhile reviewing the basic requirements of Dreamweaver when designing and then deploying static HTML pages for a web site. The following background information will be built upon when discussing the visual design of dynamic WebHub pages.

We will be considering the following HTML design entities:

HTML or XHTML web page source files	*.html
External Cascading Style Sheet files	*.css
External Javascript files	*.js
Image files	*.jpg,*.png,*.gif etc

During the development life cycle of a project, these static files generally exist in 2 places -

i) on the development computer

ii) on the production web server (and sometimes a staging server)

The exact location of files on each system is a matter of preference. In the following sections, a best-practices approach is suggested. The outcome of following these best-practices is that you will achieve:

- an easy learning curve from the design of static pages to the design of dynamic pages
- identical file content for local and posted (remote) copies of all files i.e. there will be no need to make any changes within the html / xhtml code (e.g. to links) before posting to a production server.

Firstly, we present a sample static file and some definitions that will be referred to in sections that follow.

Sample static file default.html

FileName: default.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
      <title>Document title</title>
      <meta name="description" content="Page description goes here" />
      <link rel="stylesheet" type="text/css" href="css/mystyles.css" />
      <script type="text/javascript" src="js/myscripts.js" />
</head>
<body>

Textual and graphic contents of the document.
<img src="images/image1.png" alt="image 1 - note no leading slash" />
<img src="/images/image2.png" alt="image 2 - note the leading slash" />
<a href="contact.html">Contact Page</a>

</body>
</html>
```

Resource Definitions

The following definitions are, for simplicity, restricted to the protocol http (or https) as applied to accessing both web pages and common web page resources via a WebServer.

Resource:	Any file or response that can be accessed over http(s) by using a Uniform Resource Locator (URL)
Resource Location:	This term is used to mean the **URL** of a web page (such as html, php) or an image file (such as jpeg, png) or stylesheet file (css) or other resource (e.g. js,swf) as referenced in `href=""` and `src=""` attributes. The URL always includes a path or implied path to the resource.
Resource Path	That part of a Resource Location (URL) that is the Path (implied, relative or absolute) to a resource. This defines where a resource (referenced within a web page) is located.

Resource Path Type:

There are 3 types of resource path types:

Absolute	e.g. http://www.href.com/images/mylogo.png
Document relative	e.g. images/mylogo.png
Site Root relative	e.g. /images/mylogo.png

This user guide uses the abbreviations A, D and S for these types.

A = **A**bsolute resource path

This is **the complete URL of the referenced resource** and includes the protocol. It usually starts with **http(s)://** and therefore retrieves a resource from the domain named in the URL. The named domain may be a Web Site on the same WebServer, or it may be hosted on another server.

S = **S**ite Root relative resource path

A Site Root relative path **begins with a leading forward slash '/'**, which stands for the site root folder. Because of the '/', a browser retrieves a resource relative to the Site Root of the currently displayed document (or from the domain in the <base href tag> if one exists)

D = **D**ocument relative resource path

In a Document relative path there is **no leading forward slash '/'**, therefore the reference is relative to the currently displayed document or a <base href...> tag contained within the head tag of the html code. The basic idea is to omit the part of the absolute URL that is the same for both the current document and the linked resource, providing only the portion of the path that differs.

Using Dreamweaver to design static HTML pages

Firstly we will consider the design of static html pages *without* the use of a local web server, and then discuss designing *with* the use of a local web server as a transition to the discussion of dynamic page design.

Designing static pages without local web server software

A common approach for developers of *static only* sites is as follows:

File Locations:

Development computer

i) create a separate project-folder (e.g. Project1) for each website project

ii) create a folder under the project-folder to contain files and folders that will be posted to the production server (e.g. **WebRoot)**

Example:

```
c:\Clients
        \MyClient                          (client-folder)
            \Project1                      (project-folder)
                \WebRoot                   (Dreamweaver Site Root)
                    default.html
                    contact.html
                    \css
                        mystyles.css
                    \images
                        image1.png
                        image2.png
                    \js
                        myscripts.js
                \Other
                    specs.doc
```

The folder **WebRoot** is used to contain files and folders that will be posted to the production server and also allows for the creation of other folders under the main project-folder to contain private development files such as specifications e.g. \Other\specs.doc.

Production Server

i) copy all files and folders contained **within** the development computer's **WebRoot** folder to the production server's web-root

Dreamweaver Settings:

i) create a Dreamweaver Site for each project where the Dreamweaver-Site-Root equals the on-disk folder named **WebRoot** under the project-folder.

ii) accept Dreamweaver's defaults for the Site's Testing Server, leaving *Server Model*: None, *Access*: None.

Preview in Browser analysis for viewing default.html (without a local web server)

Now, in order to discuss how the browser resolves resource references during **File** | **Preview in Browser**, we consider the sample static file default.html, Dreamweaver's settings and file locations.

Project-Folder:	c:\Clients*MyClient**Project1*
Dreamweaver Site-Root :	c:\Clients*MyClient**Project1***WebRoot**
Testing Server folder:	N/A
Testing Server URL Prefix:	N/A
Production server site root	c:\WebSites*Project1***WebRoot**
For default.html	
On-disk location	c:\Clients*MyClient**Project1*\WebRoot\default.html
Preview URL for viewing	c:\Clients*MyClient**Project1*\WebRoot\default.html
Resource reference types	mostly Document relative (D)

resource references in html	type	resolved resource locations used by a browser
`href="css/mystyles.css"`	D	c:\Clients\MyClient\Project1\WebRoot\css\mystyles.css
`src="js/myscripts.js"`	D	c:\Clients\MyClient\Project1\WebRoot\js\myscripts.js
`src="images/image1.png"`	D	c:\Clients\MyClient\Project1\WebRoot\images\image1.png
`src="/images/image2.png"`	S	c:\Clients\MyClient\Project1\WebRoot\images\image2.png
`href="contact.html"`	D	c:\Clients\MyClient\Project1\WebRoot\contact.html

Notice that the Site Root relative reference for image2.png (`src="/images/image2.png"`) is still resolved to the correct physical location because there is no local HTTP Server software involved and the browser is previewing directly from files on-disk.

Designing a static site with a local web server

Previewing web page designs in Dreamweaver can be achieved using http and a local **web server**.

Though not necessary for purely static sites, a web server **is** required when previewing dynamic sites. So consider this section to be an introduction to the upcoming methodology of file locations for dynamic site web page design.

> Note that when accessing files using a local WebServer, it is common to use “localhost”.
> e.g. http://localhost/default.html

When previewing pages via a web server, in order to access pages within **separate** web site projects, the on-disk location of the **root-folder** for each project must unique and separate. This can be achieved by either using or not using virtual directory mappings in the web server.

Before considering each approach, it is worthwhile reviewing how web-server software will resolve the location of a static web page and resources (See **Appendix G**)

The approach without using virtual directories (and this is not recommended) is detailed in **Appendix F** for completeness.

The recommended approach is to separate web sites using virtual directories.

Specifically, the *best-practices* approach is, for each separate project:

- to use a single folder that contains all files associated with that project (Web Site, documentation, graphics source etc), and
- to use a virtual directory within the HTTP Server software to map to that part of the on-disk folder structure that contains the Web Site files.

Benefits of this approach are as follows:

1. an easy transition to the design of dynamic sites by becoming familiar with
   i) previewing page designs through http://localhost/ instead of direct file access
   ii) a folder structure that is easy to adapt to that required for dynamic sites
   iii) configuring and leveraging virtual directories
2. an easy design of multiple projects **even** on HTTP Server software such as IIS 5.1 on Windows XP Pro which attempts to restrict you to working on a single web site.
3. an easy backup of all files associated with a project

A Best Practices example follows.

Example: Static site with preview via a local web server using virtual directories

File Locations:

Development computer

i) use an umbrella-folder in the root of your data drive named say **Clients**

ii) if designing for multiple clients, create a separate client-folder for your client under the umbrella-folder e.g. **MyClient**

iii) create a project-folder for the web site project, either under the related client-folder, or under the umbrella-folder (if not using client folders). e.g. **Project1Name**

iv) create a folder called **WebRoot** under the web site project-folder. This is conceptually similar to the c:\InetPub\wwwroot folder, the difference being that its child folders must be made explicitly visible using the virtual directory technique.

v) create a folder under under the WebRoot folder using an *abbreviation of the project name* e.g. **Project1Abbrev** (WebHub users note: do not make this the same as any AppID running on the development or production system)

vi) create a web server **virtual directory** named *Project1Abbrev* and map this to c:\Clients*MyClient**Project1Name*\WebRoot\ *Project1Abbrev*

Example:

```
c:\Clients                          (umbrella folder for all clients and client's projects)
    \MyClient                       (specific client folder)
        \Project1Name               (project-folder)
            \WebRoot                (Dreamweaver Site Root)
                \Project1Abbrev     (mapped to by virtual directory Project1Abbrev)
                    default.html
                    contact.html
                    \css
                        mystyles.css
                    \images
                        image1.png
                        image2.png
                    \js
                        myscripts.js
            \Other
                specs.doc
```

Production Server

i) copy all files and folders contained **within** the development computer's **Project1Abbrev** folder to the production server's web-root e.g. c:\WebSites*Project1Name*\WebRoot**Project1Abbrev**

Dreamweaver Settings:

i) create a **Dreamweaver Site** for the project and set the Dreamweaver Site Root equal to the project's **WebRoot** folder c:\Clients*MyClient**Project1Name*\WebRoot

ii) configure the site's **Testing Server** as follows:

Server Model:	None
Access:	Local/Network
Testing Server Folder:	c:\Clients*MyClient**Project1Name*\WebRoot\
URL Prefix:	http://localhost/

Note: Dreamweaver is totally oriented to the Dreamweaver Site Root so if you are running a local web-server and using this for **previewing**, then set:

Testing Server Folder = Dreamweaver Site Root

and

Testing Server URL Prefix = http://localhost.

So, for example, consider **previewing** the file default.html

i.e. c:\Clients*MyClient**ProjectName1*\WebRoot*Project1Abbrev*\default.html

with Dreamweaver settings as above.

Dreamweaver would firstly notice that this file in in the folder *Project1Abbrev*, then it would calculate the location of this folder relative to the Dreamweaver Site Root.

In this case the Dreamweaver Site Root is,
c:\Clients*MyClient**Project1Name*\WebRoot

so the path of default.html relative to the site-root is just
Project1Abbrev\default.html.

Dreamweaver would then append the **testing server URL Prefix** with this **relative path**
http://localhost + /*Project1Abbrev*/default.html

and send the following URL to the browser for preview:

http://localhost/Project1Abbrev/default.html

Local Web Server Settings – Virtual Directory:

For the browser to access the files for the project in this example:

i) a *virtual directory* named **Project1Abbrev** must exist for the web site that serves http://localhost, and

ii) this *virtual directory* must have a local path of c:\Clients*MyClient**Project1*\WebRoot\ **Project1Abbrev**

Preview in Browser analysis for viewing default.html (with a local web server)

Once again, in order to discuss how the browser resolves resource references during **File | Preview in Browser**, we consider the sample static file default.html, Dreamweaver's settings and file locations,

So far we have:

request:	http://localhost/Project1Abbrev/default.html
root-folder location:	c:\Clients\MyClient\Project1Name**WebRoot**\
resource location:	c:\Clients\MyClient\Project1Name\WebRoot**Project1Abbrev**\default.html
virtual directory:	**Project1Abbrev**
virtual dir maps to:	c:\Clients\MyClient\Project1Name\WebRoot**Project1Abbrev**

For default.html

Preview URL for viewing:	http://localhost/Project1Abbrev/default.html
On-disk location:	c:\Clients\MyClient\Project1Name\WebRoot\ Project1Abbrev\default.html
Resource reference types:	mostly Document relative (D)

Project-Folder: c:\Clients*MyClient**Project1Name*

Dreamweaver Settings

Local root folder:	c:\Clients*MyClient**Project1Name***WebRoot**
Testing Server Folder:	c:\Clients*MyClient**Project1Name***WebRoot**\
Testing Server URL prefix:	http://localhost

resource references in html	type	resolved-resource-locations used by browser
`href="css/mystyles.css"`	D	http://localhost/Proj1Abbrev/css/mystyles.css
`src="js/myscripts.js"`	D	http://localhost/Proj1Abbrev/js/myscripts.js
`src="images/image1.png"`	D	http://localhost/Proj1Abbrev/images/image1.png
`src="/images/image2.png"`	S	http://localhost/images/image2.png (resulting in an error)
`href="contact.html"`	D	http://localhost/Proj1Abbrev/contact.html

Notice that the Site Root relative reference for image2.png (`src="/images/image2.png"`) is resolved to be relative to the web-root of localhost, not relative to the current document location. The web-server would look for the file c:\InetPub\wwwroot\images\image2.png and return an error when not found. This is because the resource reference type of `src="/images/image2.png"` is Site Root relative due to the leading forward slash '/'. Note that this same reference resolved correctly when previewing without a web server (i.e. directly form the files on disk).

Designing Dynamic WebHub Page Content

The *WebHub Designer* was written to leverage the visual rendering of Dreamweaver in Design-view, but it is also a great editor for pure code.

In fact, when designing dynamic WebHub pages with Dreamweaver, you have a choice of two modes.

One is `designmode="code"` and the other is `designmode="visual"`.

Code mode

When the designmode is set to **`code`**, all features of the WebHub Designer are available except for being able to view dynamic content in Design-view.

In this mode, to preview page content, you would still require a running WebHub Application EXE and either direct access with a browser (initiated outside Dreamweaver), or you would use **File** | **Preview in Browser** (if a Dreamweaver Site and the Testing Server are correctly configured).

Visual mode

On the other hand, setting the designmode to **`visual`** enables you to view dynamic content in Design-view, in addition to providing all the features of designmode="code". This mode requires a running WebHub Application EXE, a *DynSource* configured to access this WebHubApp, and a correctly configured Dreamweaver Site. **Preview in Browser** also requires a correctly configured Testing Server for the Dreamweaver Site.

Both modes

Both modes (**`code`** and **`visual`**) require a careful consideration of folder structure and file placement, especially with regard to the location of static resources such as css, javascript and image files. We are going to maintain a best-practices approach which results in being able to use a **single set of files** to view dynamic content in Design-view, to preview page content in a browser, and to post files to a practice or production server. In other words, the file content will remain identical during all three tasks.

Before presenting a best-practices approach to folder structure and to Dreamweaver configuration issues when designing WebHub pages, we will first cover some important WebHub Application and page design fundamentals.

WebHub Applications

WebHub page content can only be viewed in a browser or in Dreamweaver Design-view when that page content is served by a running WebHub Application.

In a simplistic sense, a WebHub Application consists of:

- a compiled and running WebHub Application EXE (WebHubApp)
- an **XML file** that contains configuration and default settings for this EXE
- one of more **whteko** files that contain the WebHub page content

For each **whteko** file, there needs to be an entry in the **XML file** so that the EXE knows to load that particular whteko file's contents.

e.g. for the WebHubApp **Demo.EXE**, and the **whteko** files named **demo1.whteko** and **demo2.whteko**, the XML file **WHAppConfig_demo.xml** would contain at least the following section:

WHAppConfig_demo.xml

```
<WebHubAppConfiguration>
  <Application>
     <TekoFiles>
       <File filename="demo1.whteko" />
       <File filename="demo2.whteko" />
     </TekoFiles>
  </Application>
</WebHubAppConfiguration>
```

In addition to other content, whteko files define the PageIDs of WebHub pages within an application. The content of each WebHub page is accessed in a browser by referring to its PageID using a specially formatted URL - a WebHub URL.

WebHub URLs

WebHub URLs differ from all other static and dynamic web page URLs in that there is no unique WebHub-specific file extension present.

The URL format is:

http(s)://**domain**/**runnerpath**/**runner**?**AppID**:**PageID**[:**Session**:**parameters**]

where [ ] means optional.

e.g. for a running WebHub Application with an AppID=demo, to access a WebHub page named Home, the URL might be

http://localhost/scripts/runisa.dll?demo:Home

In this example

domain	= localhost
runnerpath	= /scripts/
runner	= runisa.dll
AppID	= demo
PageID	= Home

The runnerpath must be a path to a valid WebHub runner. When used with Microsoft IIS, the runner is an ISAPI Extension DLL named runisa.dll by default.

WebHub Sessions and Save-State

The WebHub URL format just presented contained an optional portion [] with the syntax [:Session:parameters]. Session is short for WebHub-Session-Number and this is used by WebHub to identify a unique visitor Session and to maintain State between a series of requests from a browser. The Session is also used when WebHub commands generate dynamic links and html form action targets.

There are a few main ways to Save-State between requests.

- Use a *Cookie*

  here a specifically named cookie with a unique value is created by a dynamic application upon receiving the first request from a browser. A first request is simply one that does not contain a cookie of the specific name. This cookie is then used as an identifying token for each subsequent request sent to the dynamic application by the browser. All session variables associated with this unique cookie value are stored on the WebServer until the session times out.

- Use *Hidden fields (details not covered here)*
- Use a URL based *Session Identifier* (such as a Session Number)

  With this technique, each link to a dynamic page within an application includes a *Session Identifier* in the URL. The *Session Identifier* is created by a dynamic application upon receiving the first request from a browser. (A first request is just one that does not contain a *Session Identifier* in the URL of the request). This *Session Identifier* in the URL is then used an an unique identifier for each subsequent request sent to the dynamic application by the browser. All session variables associated with this unique *Session Identifier* are stored on the WebServer until the session times out.

WebHub was one of the earliest adopters of URL based Session Identifiers (since 1995). This is implemented in WebHub as a **Session Number** (an integer usually from 6 to 9 digits) where the WebHub Connection Layer is configured to the exact number of digits required for all applications running on the same server.

Dreamweaver access to a WebHub Application also requires a session number, but the number of digits used (typically 4) is always less than that set in the WebHub Connection Layer and is configured manually for a particular WebHubApp EXE. For security reasons, the Dreamweaver session number will therefore not bump into regular visitor session numbers, and is not public.

WebHub Hyperlinks and Form Actions

To stay within an active WebHub Session while navigating around the pages of an application (i.e. to have the Save-State mechanism work throughout a series of requests from a browser), each dynamic resource location referred to on a web page must include the WebHub Session Number. This includes all hyperlinks, form action URLs.

We will explain how to do this momentarily.

WebHub Hyperlinks

In a static html file, one might have

```
<a href="contact.html">Contact Us</a>
```

On a WebHub page, this hyperlink must be generated using a WebHub Command such as JUMP. (the commands GO and HIDE would also be suitable depending on the context)

e.g. `(~JUMP|contact|Contact Us~)`

(the command format used here is `(~JUMP|PageID|Visible Phrase~)`. The AppID is implied.)

This is a WebHub Expression (in this case just a single WebHub Command) and is expanded when the page is rendered by the WebHubApp. Assuming the WebHubApp AppID is 'demo' and that the active Session Number is 453278, then the expansion would be as follows:

`(~JUMP|contact|Contact Us~)` would be replaced by

```
<a href="/scripts/runisa.dll?demo:contact:453278">Contact Us</a>
```

By using the WebHub Expression with the JUMP command, you make the WebHubApp EXE calculate the <a href_tag> dynamically, so the current site visitor's session number is filled in automatically, every time (without using any cookies).

WebHub Form Actions

In a static html file, one might have

```
<form method="POST" action="postpage.html">
```

In a WebHub page, the **value** of the `action` attribute must be generated by a WebHub Command and this is most commonly the ACTION command.

```
<form method="POST" action="(~ACTION|PageID~)">
```

Assuming the same AppID and session number as above, this line, when expanded by the WebHubApp would become:

```
<form method="POST" action="/scripts/runisa.dll?demo:postpage:453278">
```

whteko Files – an introduction

WebHub page content is defined and contained within a **whteko** file. More than one WebHub page can exist in a single whteko file, but for the moment we will consider a single page per file.

We will start with the simplest of whteko files and build up from there.

Example 1: toosimple.whteko containing one WebHub page named "toosimple"

toosimple.whteko

```
<whteko>
<whpage pageid="toosimple">
Hello World. The time is (~CentralInfo.WebTimeGMT~)
</whpage>
</whteko>
```

Example 1 shows the simplest of WebHub pages in the most basic whteko file structure. Though not a valid html page, the contents of the WebHub page `toosimple` will still render in a browser and the WebHub Expression `(~CentralInfo.WebTimeGMT~)` will be expanded by the WebApp to show the date and time on the web server on which the WebApp is running.

So, if we assume a locally running WebHub Application (with AppID=demo) with an entry in its XML file of

```
<TekoFiles>
  <File Filename=toosimple.whteko />
</TekoFiles>
```

then requesting http://localhost/scripts/runisa.dll?demo:toosimple

will result in a browser displaying (where the date and time are just an example)

Hello World. The time is Tue 08 Mar 2006 05:00:12 GMT

This example serves to illustrate some basic requirements of whteko files:

All whteko files must contain the top level **<whteko_tag>**

WebHub pages are defined within a **<whpage_tag>** which is always a direct child tag of the **<whteko_tag>**

The **<whpage_tag>** must have an attribute defining the name of the WebHub page **pageid="webhubpagename".**

Warning: All <wh... tags **must be positioned flush left** for the current release of WebHub. Do not indent these tags **!**

To apply **File** | **Preview in Browser** to a WebHub page from Dreamweaver requires that a Dreamweaver Site has been configured. This will be discussed later.

For now, we will consider how to display the content of the example page named `toosimple` in Dreamweaver Design-view.

Displaying WebHub dynamic page content in Dreamweaver Design-view

The default design mode of a whteko file is `designmode="code"`. This is because continually requesting dynamic content from a running WebHubApp is costly in terms of time, so unless specifically required, `designmode="visual"` is not used by default.

So if the above example file (**toosimple.whteko**) were to be loaded into Dreamweaver and the view mode was switched from Code-view to Design-view, the only content to display would be

Hello World. The time is (~CentralInfo.WebTimeGMT~)

This is due default behavior by the WebHub Designer that is triggered by default settings of two **<whteko_tag>** attributes, `designpage` and `designmode`.

The default `designpage` value is the **first** WebHub page to be defined in a **whteko** file. From Example 1, the first (and only) page definition was in the line **<whpage pageid="toosimple">** so by default for the example whteko file we have **designpage="toosimple"**.

The default `designmode` value is `code` so by default **designmode="code"**.

For clarity, Example 1 is rewritten here with the implicit values of these whteko tag attributes made explicit.

Example 2:

```
<whteko
      designpage="toosimple"
      designmode="code"
>

<whpage pageid="toosimple">
Hello World. The time is (~CentralInfo.WebTimeGMT~)
</whpage>

</whteko>
```

When **designmode="code"**, the display of dynamic content in Design-view is disabled and only the Code-view features of the WebHub Designer are active. This `code` mode is very useful when visual design is not required because by removing the need to call a WebHubApp to expand dynamic content, a number of aspects of editing the whteko code are considerably faster.

Now let's consider how to display our toosimple page visually!

Visual display of dynamic WebHub page content in Dreamweaver requires the following:

- a running WebHub Application EXE (WebHubApp)
- a properly configured *DynSource* pointing to that WebHubApp
- correct setting of the **<whteko_tag>** attributes `designdynsrc`, `designpage`, `designmode`

  `designdynsrc="aValidDynSource"` (this points to the correct WebHubApp)

`designpage="PageIDOfPageToView"` (this PageID must be in the whteko file)

`designmode="visual"` (this enables viewing of dynamic content)

Thus, to display the dynamic content of the example page named `toosimple` in Dreamweaver Design-view, we require:

- a running WebHub Application EXE

  assume we have a locally running WebHub Application with the following configuration:

AppID	demo
remote-design PageID	RemoteDesign
remote-refresh PageID	RemoteRefresh
Dreamweaver Session	1214
access URL	http://localhost/scripts/runisa.dll?demo

- a properly configured DynSource pointing to that EXE

  assume we have a DynSource saved to the Dynamic Content Source Configuration as

  ```
  demo-practice=http://localhost/scripts/runisa.dll?demo|RemoteDesign|RemoteRefresh|1214|(~
  ```

- correct setting of the **<whteko_tag>** attributes `designdynsrc`, `designpage`, `designmode`

  the required settings would be as follows:

  `designdynsrc="demo-practice"` (this points to the correct WebApp EXE)
  `designpage="toosimple"` (this is the PageID to view and is in the whteko file)
  `designmode="visual"` (this enables viewing of dynamic content)

So, to enable viewing the page toosimple in Design-view, our example would become:

Example 3:

```
<whteko
      designdynsrc="demo-practice"
      designpage="toosimple"
      designmode="visual">

<whpage pageid="toosimple">
Hello World. The time is (~CentralInfo.WebTimeGMT~)
</whpage>
</whteko>
```

Now when switching from Code-view to Design-view, the content displayed would be (where the date and time are an example only)

Hello World. The time is Tue 08 Mar 2006 05:00:12 GMT

When switching to Design-view (for the first time or after a change has been made to the code),the WebHub Designer sends all WebHub Expressions contained in the designpage to the WebHubApp pointed to by the *DynSource* and receives back the expanded dynamic content for each. This content is then merged this with unaltered static content in the page, and the result is displayed in Design-view. Dynamic content regions are locked against editing, but render fully as they would if the page were to be viewed in a browser.

In this example there was only one WebHub Expression `(~CentralInfo.WebTimeGMT~)` and this was expanded as `(~CentralInfo.WebTimeGMT~)` -> **Tue 08 Mar 2006 05:00:12 GMT**

Next up, we transform the example page content for default.html into a WebHub whteko format.

Before doing so, however, a full explanation of how to treat references to static resources is presented.

References to stylesheet and graphic resources

One of the most difficult of aspects of designing WebHub pages with Dreamweaver is the approach to static resource references within the html code. Dreamweaver's Design-view requires that all static resources (e.g. images and css files) are on the local development computer AND that they are referenced as **relative** to the document being edited. WebHub, on the other hand, requires **absolute or site-relative references** that are relative to the web-root of the web-server so that a browser can correctly resolve and request static resources that are sensibly located on the production server.

Because of the nature of a WebHub URL, relative resource references would be relative to the location of the runisa.dll file.

e.g.

Suppose runisa.dll is located on the Production Server as c:\InetPub\wwwroot\scripts\runisa.dll

Next, consider a request for the WebHub page named `default`

http://localhost/scripts/runisa.dll?demo:default

If the page `default` were to have a static resource reference to an image say `image1.png` such as `<img src="image/image1.png">`, then the browser would resolve the location of this image file to be

http://localhost/scripts/image/image1.png

This is not considered a best-practices approach to image location on the production server.

Part of the suggested approach will be to always use site-relative or absolute references to static resources.

So the above example image tag would become

`<img src="/image/image1.png">` (note the leading `'/'`)

and the browser would request

http://localhost/image/image1.png

where `image` was defined as a **virtual directory**.

The details of this will be covered more fully later, but for now it is important to note the conflicting needs of Dreamweaver Design-view and browser-viewing with respect to static resource references in a WebHub page.

To resolve this conflict, all static resource references on WebHub pages are expressed using an appropriate WebHub macro that will expand to the correct value for Dreamweaver Design-view, and also expand to the correct value for browser-viewing and final production.

The technique is to define a few application-wide macros – one for use with stylesheets, one for graphics and one for javascript as a minimum. All 3 share a common resource root which is itself a macro that handles whether or not you are using Dreamweaver or viewing via a Browser.

Using best practices to reference stylesheet and graphics files in WebHub pages

Assume that you have resources stored locally under the following folders:

stylesheets	Dreamweaver Site Root/Live/WebHub/Project1Abbrev1/css
javascript	Dreamweaver Site Root/Live/WebHub/Project1Abbrev1/js
images	Dreamweaver Site Root/Live/WebHub/Project1Abbrev1/images

In your html code you might have:

```
<link rel="stylesheet" type="text/css" href="(~mcCSSRoot~)mystyles.css"
/>
<script type="text/javascript" src="(~mcJSRoot~)myscripts.js" />
<img src="(~mcImageRoot~)image1.png" />
```

In Dreamweaver Design-view, you would want the WebHub Expressions to expand as follows

```
(~mcCSSRoot~)    -> /Live/WebHub/Project1Abbrev1/css
(~mcJSRoot~)     -> /Live/WebHub/Project1Abbrev1/js
(~mcImageRoot~)  -> /Live/WebHub/Project1Abbrev1/images
```

But in Browser-Preview and Browser-viewing in general, you may want the WebHub Expressions to expand this way:

```
(~mcCSSRoot~)    -> /Project1Abbrev/css
(~mcJSRoot~)     -> /Project1Abbrev/js
(~mcImageRoot~)  -> /Project1Abbrev/images
```

To achieve this, in the application XML file you might have:

WHAppConfig_demo.xml

```
<AppSettings>
  <AppSetting name="ResourceFolder" value="Project1Abbrev/" />
</AppSettings>
```

And in one of the whteko files for the application you might have:

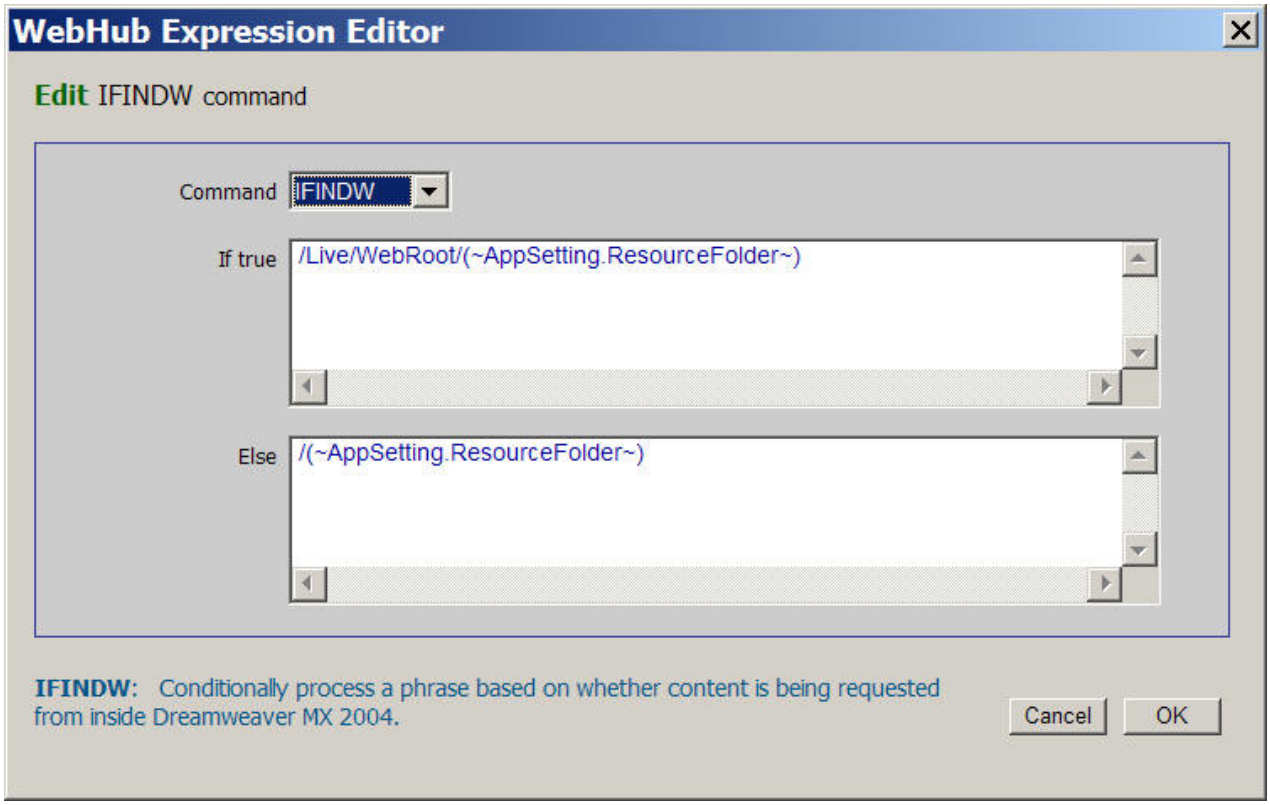
```
<whteko ...>
...
<whmacros>
mcResourceRoot=(~IFINDW|/Live/WebRoot/(~AppSetting.ResourceFolder~)||
/(~AppSetting.ResourceFolder~)~)
mcCSSRoot=(~mcResourceRoot~)css/
mcJSRoot=(~mcResourceRoot~)js/
mcImageRoot=(~mcResourceRoot~)images/
</whmacros>
...
</whteko>
```

IFINDW is a WebHub Command that has the syntax

```
(~IFINDW|truePhrase||falsePhrase~)
```

When this expression is expanded, if in Dreamweaver the whole expression is replaced by truePhrase, otherwise it is replaced by falsePhrase. For Example:

Figure 6-1 WebHub Expression Editor and example IFINDW command



[Figure 6-1] shows and example of an IFINDW command being edited in the WebHub Expression Editor.

If calculated for Dreamweaver Design-view, the WebHub Expression

```
(~IFINDW|/Live/WebRoot/(~AppSetting.ResourceFolder~)||
/(~AppSetting.ResourceFolder~)~)
```

would be replaced by

```
/Live/WebRoot/(~AppSetting.ResourceFolder~)
```

else it would be replaced by

```
(~AppSetting.ResourceFolder~)
```

Remaining nested WebHub expressions would be further expanded until no expressions remained.

The full expansion of these macros is illustrated in the following tables.

Tip: The best method for definitively checking the result of a given WebHub Expression expansion is to request the WebHub page in a browser then View Source. Look in the html code for where the Expression was positioned in the original source code and examine the resulting output.

In Dreamweaver Design-view, the resource references would expand to be relative to the Dreamweaver Site Root:

Expression	*Initial Expansion*	*Final Expansion*
(~mcResourceRoot~)	/Live/WebRoot/(~AppSetting.ResourceFolder~)/	/Live/WebRoot/Project1Abbrev/
(~mcCSSRoot~)	(~mcResourceRoot~)css/	/Live/WebRoot/Project1Abbrev/css/
(~mcJSRoot~)	(~mcResourceRoot~)js/	/Live/WebRoot/Project1Abbrev/js/
(~mcImageRoot~)	(~mcResourceRoot~)images/	/Live/WebRoot/Project1Abbrev/images/
href="(~mcCSSRoot~)mystyles.css"		href="/Live/WebRoot/Project1Abbrev/css/mystyles.css"
src="(~mcJSRoot~)myscripts.js"		src="/Live/WebRoot/Project1Abbrev/js/myscripts.js"
src="(~mcImageRoot~)image1.png"		src="/Live/WebRoot/Project1Abbrev/images/image1.png"

In Browser-viewing, the resource references would expand to be relative to the web-root of the WebServer:

Expression	*Initial Expansion*	*Final Expansion*
(~mcResourceRoot~)	/(~AppSetting.ResourceFolder~)/	/Project1Abbrev/
(~mcCSSRoot~)	(~mcResourceRoot~)css/	/Project1Abbrev/css/
(~mcJSRoot~)	(~mcResourceRoot~)js/	/Project1Abbrev/js/
(~mcImageRoot~)	(~mcResourceRoot~)images/	/Project1Abbrev/images/
href="(~mcCSSRoot~)mystyles.css"		href="/Project1Abbrev/css/mystyles.css"
src="(~mcJSRoot~)myscripts.js"		src="/Project1Abbrev/css/myscripts.js"
src="(~mcImageRoot~)image1.png"		src="/Project1Abbrev/images/image1.png"

The DOCTYPE Tag and Validation

DOCTYPES are essential to the proper rendering and functioning of web documents in compliant browsers. Appendix H **– DOCTYPES that work**, lists the common DOCTYPE tags that work and use valid DTDs as published by the W3C.

Although any valid DOCTYPE can be placed at the start of a WebHub page's content, if you want to use Dreamweaver's **File** | **Check Page** | **Validate as XML** on a **whteko** file, then you will need to use a special approach using one of two custom WebHub DOCTYPE tags.

These custom WebHub DOCTYPE tags enable full validation of **whteko** file contents, automatically including xhtml validation of all contained html code.

WebHub currently supports whteko validation where your page content is compliant with either **XHTML 1.0 Transitional** or **XHTML 1.0 Strict**. We recommend designing to the XHTML 1.0 Strict standard if possible.

For validation to work correctly, you will need to use one of the two special DOCTYPE tags at the very start of your whteko file, and then include the corresponding W3C DOCTYPE tag at the very start of page content for each WebHub page defined in the whteko file. Importantly, the W3C DOCTYPE tag will need to be included within a `<![CDATA[ ]]>` declaration.

The two special WebHub whteko DOCTYPE declarations are

```
<!DOCTYPE whteko PUBLIC "-//HREF//DTD whteko stage 2.14//Strict//EN//"
"http://webhub.com/dtd/0214/whteko.dtd">
```

and

```
<!DOCTYPE whteko PUBLIC "-//HREF//DTD whteko stage
2.14//Transitional//EN//"
"http://webhub.com/dtd/0214/transitional/whteko.dtd">
```

One of these WebHub DOCTYPE tags must be placed before the opening **<whteko_tag>**

Example:

sample.whteko file

```
<!DOCTYPE whteko PUBLIC "-//HREF//DTD whteko stage 2.14//Strict//EN//"
"http://webhub.com/dtd/0214/whteko.dtd">
<whteko
  designdynsrc="demo-practice"
  designpage="default"
  designmode="visual">

<whpage pageid="default">
<whoutput>
<![CDATA[<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">]]>
<html>
<head>
  <title>Document title</title>
and so on
```

See **8. Appendices:** – **Appendix I – Advanced Tips** for a better approach to including the <!DOCTYPE tag at the start of each WebHub page.

Converting the example default.html to a WebHub page

We are now going to take the example default.html static file from a previous section and convert it to a WebHub page. This conversion will serve to highlight a number of aspects of designing WebHub pages using Dreamweaver, including those covered above.

default.html (from before)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
      <title>Document title</title>
      <meta name="description" content="Page description goes here" />
      <link rel="stylesheet" type="text/css" href="css/mystyles.css" />
      <script type="text/javascript" src="js/myscripts.js" />
</head>
<body>
Textual and graphic contents of the document.
<img src="images/image1.png" alt="image 1 - no leading slash" />
<img src="/images/image2.png" alt="image 2 - with leading slash" />
<a href="contact.html">Contact Page</a>
</body>
</html>
```

default.whteko (WebHub version of the default.html page)

```
<!DOCTYPE whteko PUBLIC "-//HREF//DTD whteko stage 2.14//Strict//EN//"
"http://webhub.com/dtd/0214/whteko.dtd">
<whteko
  designdynsrc="demo-practice"
  designpage="default"
  designmode="visual"
  notes="default.html converted into a WHTEKO file">

<whpage pageid="default">
<whoutput>
<![CDATA[<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">]]>
<html>
<head>
  <title>Document title</title>
  <meta name="description" content="Page description goes here" />
  <link rel="stylesheet" type="text/css" href="(~mcCSSRoot~)mystyles.css"
/>
  <script type="text/javascript" src="(~mcJSSRoot~)myscripts.js" />
</head>
<body>
Textual and graphic contents of the document.
<img src="(~mcImageRoot~)image1.png" alt="image 1" />
<img src="(~mcImageRoot~)image2.png" alt="image 2" />
(~JUMP|contact|Contact Page~)

</body>
</html>
</whoutput>
</whpage>

</whteko>
```

Notes about the conversion of default.html into default.whteko

The entire default.whteko file will validate when using Dreamweaver's **File** | **Check Page** | **Validate as XML** feature.

The WebHub page named `default` is defined within the tag **`<whpage pageid="default">`**

The part of the WebHub page that will be sent to the browser is contained within a starting `<whoutput>` tag and ending `</whoutput>` tag. The use of the `<whoutput_tag>` is not necessary but is recommended best-practices.

The W3C DOCTYPE tag that forms part of the final web page output is included within a `<![CDATA[ ]]>` declaration so that the overall whteko file can be validated. The recommended technique for including a DOCTYPE tag on each WebHub page in your project is detailed in **8. Appendices:** – **Appendix I – Advanced Tips - Using a macro to include the <!DOCTYPE tag at the start of each WebHub page.**

Folder Layout and File Location for WebHub Applications

The following example uses a best-practices approach for the location of all files associated with a WebHub Application on both the development computer and production server. Importantly, should you use a version control system (or move to one in the future), the suggested approach will help to streamline your implementation - essentially because the folder names are likely to persist for the lifetime of your project.

Initial Planning

Parent folder: **Projects** or **Clients***`MyClient`*
Project Name:* `Project1Name`*
Project Abbreviation:* `Project1Abbrev`*
AppID:* `demo` (must differ from project abbreviation)
Home PageID: `default` (name of the web page which will display if none specified)
Connectivity local to remote: ?

Preview in browser required: yes /no
If yes, do you want to override originals on the production server when previewing or use temporary copies in a separate testing folder location. We recommend the latter.

Development computer: File Locations

Dreamweaver developer tasks:

i) use an umbrella-folder in the root of your data drive e.g. **Projects** or **Clients**

ii) if designing for multiple clients, create a separate client-folder for your client under the umbrella-folder e.g. **Clients*`MyClient`***

iii) create a project-folder for the website project either under the umbrella-folder (if not using client folders) or under the relevant client-folder e.g. **Projects*`Project1Name or`* Clients*`MyClient\Project1Name`***

iv) create folders named **Live** and **Source** under the project-folder

v) create folders named **WebHub** and **WebRoot** under the website project-folder. **WebRoot** is conceptually similar to the c:\InetPub\wwwroot folder, the difference being that its child folders must be made explicitly visible using the virtual directory technique.

vi) create the folders **Apps, Library** and **whteko** under the **WebHub** folder. The application XML file and all whteko files are located under the **whteko** folder in a folder named after the AppID of the WebApp (in this case **demo**).

vii) create a folder under under the WebRoot folder using an ***abbreviation of the project name*** e.g. ***`Project1Abbrev`*** (WebHub users note: do not make this the same as any AppID running on the development or production system).

viii) Locate all static resources in their respective sub-folders of css, images, js etc. under the `Project1Abbrev` folder.

ix) create a virtual directory named `Project1Abbrev` and map this to c:\Clients\`MyClient`\`Project1Name`\Live\WebRoot\ `Project1Abbrev`

x) If supporting Dreamweaver's **Preview in Browser** feature, and you want to avoid overwriting the original whteko files during previewing, you should create a folder for the preview copies, e.g. **`c:\temp\DWPreview\`**. The name of the folder must be entered exactly under <RemoteDesignDefaultInterface> in the application's XML file. Note: there is no need to backup any files in the Preview folder.

Example:

```
c:\Clients                          ( umbrella folder for all clients projects )
    \MyClient                       ( separate folder for a specific client )
        \Project1Name               ( project-folder = Dreamweaver site root )
            \Live                   ( folder to segregate files for production sever )
                \WebHub             ( parent folder for WebHub Application Server files )
                    \Apps           ( folder for WebHub Application EXEs )
                        Demo.exe
                    \Library        ( optional folder for DLLs and Delphi packages )
                    \whteko
                        \demo       ( whteko folder for AppID = demo )
                            WHAppConfig_demo.xml
                            default.whteko
                            contact.whteko
                \WebRoot            (parent folder for static resource files)
                    \Project1Abbrev (mapped to by virtual directory named Project1Abbrev)
                        \css
                            mystyles.css
                        \images
                            image1.png
                            image2.png
                        \js
                            myscripts.js

            \Source                 (folder to segregate development only files)
                \Docs
                    specs.doc
                \whApps
                    \demo           (folder for Delphi source for Demos.exe)
                        demo.dpr
                        demo_*.pas
                        demo_*.dfm

c:\temp
      \DWPreview
```

In WHAppConfig_demo.xml:

```
<WebHubAppConfiguration>
  <Application>
    ...
    <RemoteDesignDefaultInterface>
      <TestingServerURLPrefixMappedTo value="c:\temp\DWPreview\" />
    </RemoteDesignDefaultInterface>
    ...
  </Application>
</WebHubAppConfiguration>
```

Production Server: File Locations

Network Administrator tasks

i) create a folder to hold all files for the project e.g. ***c:\WebSites\Project1Name***. This folder will be the *production server site-root* for the project and corresponds to the Dreamweaver site-root folder on the development computer.

ii) if applicable, create an ftp login or version-control login for the Dreamweaver developer, where the ftp (or version control) root is equal to the production server site-root.

iii) communicate the login details to the Dreamweaver developer

iv) under the project site-root, create a folder named **Live**

v) under **Live,** create a folder named **WebRoot** (Note: the Dreamweaver developer will create the WebHub folder and other required folders.)

vi) under **WebRoot** create a folder named **Project1Abbrev** (this must be the exact project abbreviation used by the developer for their local Dreamweaver Site-root)

vii) in the HTTP Server Software manager, create a **virtual directory** named **Project1Abbrev** pointing to the **Project1Abbrev** folder on disk. When configuring the security for the virtual root, allow read access and optionally directory browsing access, but not script access nor program execution access.

Example:

```
c:\WebSites                 (umbrella folder for all websites)
    \Project1Name           (project-folder = root for site = root for ftp or version control)
        \Live
            \WebHub                 (parent folder for WH Application Server files)
                \Apps               (folder for WebHub Application EXEs)
                    Demo.exe
                \Library            (optional)
                \whteko
                    \demo
                        WHAppConfig_demo.xml
                        default.whteko
                        contact.whteko
            \WebRoot                (parent folder for static resource files)
                \Project1Abbrev     (mapped to by virtual directory Project1Abbrev)
                    \css
                        mystyles.css
                    \images
                        image1.png
                        image2.png
                    \js
                        myscripts.js
```

Development computer: Dreamweaver Settings

Dreamweaver developer tasks:

i) create a **Dreamweaver Site** for the project and configure the site's name and root folder (on the **Local Info** tab)

Set the Dreamweaver Site Root (Local root folder) equal to the project-folder e.g.

Site Name:	*MySiteName*
Local Root Folder: c:\Clients*MyClient**Project1Name*	
	☑ Refresh local file list automatically
Default Images folder:	leave blank as not recommended
HTTP address:	leave blank. This url for the Link Checker is not required when developing WebHub Applications.

ii) Configure the site's remote server (on the **Remote info** tab)

Caveat: This task applies if and only if you can post your files to another computer using ftp, LAN, SourceSafe or WebDAV. If you are using a version control system such as CVS, which is not supported directly by Dreamweaver, you will not be able to use the remote server feature. Instead you will need to use the interface specific to your version control system to transfer files (e.g. the TortoiseCVS client for CVS).

Your access details will vary and you need to get them from the Network Administrator. We will show an example using ftp because that is the most common.

Access:	ftp	
FTP host:	ftp.webhub.com	
Host directory:	blank	
Login:	DemoDWUser	
Password:	*****	☑ Save

Leave all other settings blank or configure them according to instructions from your Network Administrator.

Click the **Test Button**. You should see the following reply (after a delay):

"*Macromedia Dreamweaver 8 connected to your Web server successfully*."

iii) Configure the site's Testing Server (on the **Testing Server** tab)

Caveat: This task applies if and only if you want to use Dreamweaver's **File | Preview in Browser** feature. Otherwise It is not required. e.g.

Server Model:	WebHub v2,
Access:	LAN/local
Testing Server Folder:	c:\temp\DWPreview
URL Prefix:	http://localhost/scripts/runisa.dll?demo:RemotePreview::

Transfer of files from development to production

Copy the files and folders contained within the **Live** folder on the development computer to the **Live** folder on the staging or production server.

* * *

This concludes the chapter dealing with Application Programming. For further information about WebHub syntax and WebHub application development in general, please refer to the various resources detailed at the start of the next chapter: **7. Support** -**Resources**.

7. Support

Resources

The following resources cover the entire syntax of WebHub in great detail:

1. WebHub Syntax Stage 2 Reference
   http://www.href.com/pub/relnotes/WebHubSyntaxStage0214.pdf

2. Quick Reference for WebHub Commands
   http://www.href.com/pub/docsnhelp/whQuickRefForCommands.html

3. Guide to Configuration Version 004
   http://www.href.com/pub/docsnhelp/WHConfiguration-cv004.pdf

And these resources provide guidance for WebHub Programmers:

WebHub Developer Manual (pending an update in Q1-2007)
http://www.href.com/whManual

WebHub Help File (relatively accurate through WebHub v2.053; pending update)
http://www.href.com/pub/docsnhelp/Help/WHUB-HLP.ZIP

WebHub Demos

There are over two dozen demos (most very simplistic) on http://demos.href.com, and all include full Delphi and WHTEKO source. All have been programmed to work with Dreamweaver. Click into any demo, and on its "welcome" page (where the PageID is pgCover), you will find a link to download files for that demo. You will also find links to view the source code over the web. In http://demos.href.com/download you can find ZIP files containing all demo code associated with a particular version of WebHub.

WebHub Programmers are strongly encouraged to read the release notes whenever upgrading. Release notes are archived at http://www.href.com/pub/relnotes

Troubleshooting

Dreamweaver appears to freeze

Dreamweaver may be unresponsive, for up to 30 seconds (or so) when it tries to retrieve dynamic content for synchronizing or viewing a WebHub page design. After the delay, an error message displays.

This is due to a timeout in the WebHub Designer that is activated when the Designer is unable to communicate (over http) with the running WebHub Application (that is pointed to by the DynSource in your document). This delay and associated error message most often occur when you are either opening or making changes to a document.

See the next section describing the error message that displays and possible causes.

Error: Dynamic Content is not available for Design View

An error message will display when the Designer is unable to communicate over http with a running WebHub Application (one that has been correctly compiled and configured to act as a DynSource).

This type of error message will display each time that you open a document, or make changes to a document, when to following situation applies:

1. a valid DynSource is specified in the designdynsrc attribute e.g. `designdynsrc="demo"`
2. the design mode is set to visual `designmode="visual"` (and sometimes `"code"`)
3. the Web Application pointed to by the designdynsrc is not available because of any or all of the following factors:
   (i) the configured DynSource URL is invalid
   (ii) the web server is down or the port is wrong
   (iii) the Hub is not running
   (iv) the WebHub Application is not running
   (v) the WebHub Application is not compiled / configured correctly because:
      - it does not contain the required Dreamweaver interface units (Delphi pas files)
      - it is not configured to allow access, either in terms of PageID or SessionNo
      - it does not contain the required Dreamweaver interface pages (in a whteko file)
      - it has a security rule which is rejecting the Dreamweaver request

If the error message occurs, then ensure that

1. the configured DynSource URL is correct (and that the port is correct)
2. the web server is running
3. the Hub is running
4. the correctly compiled and configured WebHub application is running

If the error message persists, look at the WebHub application GUI, especially the Dreamweaver Interface panel, and note the settings and interaction details when a request is sent from Dreamweaver.

For topologies where Dreamweaver is on a separate computer from the WebHubApp, WebHub Programmers can use an HTTP sniffer to determine what is sent between Dreamweaver and the WebHubApp. An HTTP sniffer can be downloaded from here: http://www.effetech.com Sometimes seeing the HTTP headers and exact document contents will explain what is happening; for example, you might see a clear LOCATION command and thereby realize that the WebHubApp is bouncing the request, probably due to a security rule.

Another way to gain information about a problematic DynSource is to test it using a web browser. Copy the DynSource definition, and turn it into a valid WebHub URL by changing | to : and removing any information about the parentils. This will help you test whether the web server is functioning, and whether the WebHubApp is available, and how it treats the session number being in the URL.

Note: The error message (preceded by a delay) can display when a document is first opened, even if not in Design view, because Dreamweaver tries to synchronize Code view and Design view at the most inconvenient moments (as far as the WebHub Designer is concerned).

Dreamweaver unexpectedly marks the current document as changed (*)

This can happen when using the WebHub Navigator panel because this panel automatically adjusts the `designpage` attribute in the **`<whteko_tag>`** :

1. when you change the Design Page with the Navigator's drop-down, or
2. when the Navigator needs to synchronize itself with the document.

Due to limitations in Dreamweaver's extension API, there is no way for the Navigator to avoid the need to make a change to the document, so please be aware of this behavior when opening a whteko file for viewing when using the Navigator.

To avoid this situation, just close the Navigator floating panel.

Technical Support

Free Support for Installation

HREF Tools Corp. provides free support for installation via its website www.href.com.

Go to http://www.href.com/GetHelp and click the Installation link.

Fill in the form and you will receive a reply by email.

Free Support for Usage and Design Issues

HREF Tools Corp provides free support for usage and design issues via the WebHub listserv.

Instructions for subscribing are at www.href.com/listserv

When you use the WebHub listserv, HREF is able to save all the questions and answers for the benefit of all users. The archive is on-line and searchable. Messages prior to August 2004 are searchable at http://support.href.com and later messages are viewable at http://lists.sonic.net/pipermail/webhub-list. If you have urgent or confidential support requirements, a Paid Support option is also available.

Paid Support

For paid support, either

email techsupport@href.com

or

go to http://www.href.com/GetHelp and click the Paid Tech Support link and fill in the form.

Your Feedback

Feedback about the contents of this User Guide is welcome. Please email techsupport@href.com with your suggestions and observations.

8. Appendices

Appendix A – Files created & changed by WebHub Designer during installation

The *WebHub Designer* extension suite consists of the following Dreamweaver Extension types:

- Command
- Custom Extension DLL
- Data Translator
- Floating Panel
- Menu Command
- Server Model
- Toolbar
- Tag-Library and Editor

A new document type (WebHub v2), document extension (whteko), ftp extension (whteko) and augmented code coloring are also included.

When the extension suite is installed using the Macromedia Extension Manager, a number of files are created (and some are changed) in the installing user's Dreamweaver Configuration Folder and subfolders.

On most Windows operating systems, the configuration folder used for installation is:

```
C:\Documents and Settings\username\Application Data\Macromedia\Dreamweaver MX 2004\Configuration
```

however on some multi-user systems such as Windows Server 2003, the installation folder may be:

```
C:\Documents and Settings\All Users\Application Data\Macromedia\Dreamweaver MX 2004\Configuration
```

Here is a list of the files that are created and changed when WebHubDWMgr.mxp is installed.

Assume all files are created unless annotated as (changed).

```
...\Configuration
        Extensions.txt                  (changed)
        FTPExtensionMap.txt             (changed)
        \CodeColoring
                WebHubCodeColoring.xml
                WebHubColors.xml
        \Commands
                whCmdCommandEditor.htm
                whCmdEditDynSrcConfig.htm
                whCmdEditExpression.htm
                whCmdEditPreferences.htm
                whCmdInitWhtekoFileType.htm
                whCmdRefreshDesignDynSrc.htm
                whCmdReloadConfig.htm
        \Content
                \Reference
                        \WebHub
                                default.htm
                                Reference.xml
                                whteko.htm
                                whTopLevelView.htm
        \DocumentTypes
```

MMDocumentTypesWH.xml
\NewDocuments
Default_wh2.whteko
\Floaters
whFloatCommands.htm
whFloatComponents.htm
whFloatMain.htm
whFloatNavigator.htm
whFloatTekoTags.htm
\JSExtension
WebHubDWMgr.dll
WebHubDWMgr.config (only created/changed when saving configuration items)
\Menus
menus.xml (changed)
\ServerModels
WebHub_2.htm
\Strings
documenttypesWH.xml
\TagLibraries
TagLibraries.vtm (changed)
\WebHub_v2
loc_stringsWH.js
stringsWH.js
TagChooser.xml
whdesign.vtm
whdoc.htm
whdoc.vtm
whdroplet.htm
whdroplet.vtm
whmacros.vtm
whoutput.vtm
whpage.htm
whpage.vtm
whpagesettings.vtm
whpagesettingslist.htm
whpagesettingslist.vtm
whprep.vtm
whremote.htm
whremote.vtm
whsketch.htm
whsketch.vtm
whteko.htm
whteko.vtm
whtranslation.htm
whtranslation.vtm
whtranslations.htm
whtranslations.vtm
\Toolbars
toolbars.xml (changed)
whToolOnIdleHandler.htm
\Translators
WebHubTranslator.htm
\WebHub
\Images
clear.gif
scrollBtnBot.png
scrollBtnBot_sel.png
scrollBtnTop.png
scrollBtnTop_sel.png
scrollThumbBot.png
scrollThumbMid.png
scrollThumbTop.png
treeAtoZ.png
treeAtoZ_sel.png
treeCategory.png
treeCategory_sel.png
treeCollapseAll.gif
treeExpandAll.gif
treeFontSelector.gif
treeIconToggle.gif
treeInsert.gif
treeInsert_dis.gif
treeInsert_sel.gif
treeMinus.png
treePlus.png
treeSelect.gif
treeSelectionDot.gif
WebHubWH.jpg

whRefreshButton.jpg
\Scripts
DreamTree.js
WebHubTranslator.js
whCmdCommandEditor.js
whCmdEditDynSrcConfig.js
whCmdEditExpression.js
whCmdEditPreferences.js
whCmdInitWhtekoFileType.js
whCmdRefreshDesignDynSrc.js
whCmdReloadConfig.js
whCommandDefinition.js
whCommon.js
whExtractExpression.js
whFloatCommands.js
whFloatComponents.js
whFloaterToolbar.js
whFloatMain.js
whFloatNavigator.js
whFloatTekoTags.js
whScrollCanvas.js
whShowHelpInReferencePanel.js
whToolOnIdleHandler.js
whTreeCommon.js
whTreeData.js
whTreeDataForCommands.js

Appendix B – WebServer Overview

Choosing HTTP Server software

Each practice and production server requires at least one HTTP server software package to be operational. This appendix gives some tips for selecting an HTTP server, and also highlights essential web server features that you need to be familiar with when using WebHub and the WebHub Designer

If you are on any of the following operating systems then you automatically have the use of Microsoft's IIS (web server software) at no additional cost:

Windows XP Professional
Windows 2000 Server
Windows Server 2003

Windows XP Home, however, no longer supports Internet Information Services (IIS) or Personal Web Server (PWS) in XP Home Edition. The official line from Microsoft is below.

> "Windows XP Home Edition does not include or support any versions (1.0, 2.0, 4.0) of Microsoft Personal Web Server (PWS). Users that need Web server functionality in a desktop operating system should use Windows XP Professional."

Nonetheless, Windows XP Home Edition is ok to use as a practice server, as long as you install a third party web server product. We recommend considering

Sambar from www.sambar.com

or if you have a strong preference for Apache then

Apache from www.apache.org

The reason that we recommend Sambar over Apache is only because Sambar supports ISAPI which is required for WebHub's runisa.dll, which is the fastest runner. WebHub's runbin.exe, which operates over cgi-bin, can be used with Apache as long as the WebHub programmer adjusts certain values within the application to support cgi-bin.

If you are using Windows XP Professional, and need to run two or more individual websites that have their own IP# (say because each needs an individual secure certificate), then you will need to run a third party web server instead of, or in addition to, IIS5.1. It is easier to replace the web server entirely so that all websites use port 80.

Essential HTTP Server Features

Despite the extensive documentation accompanying http server software, from a WebHub / WebHub Designer perspective, there are only two essential features to consider.

These are:

1. The default Web Site Document Root
2. Virtual Roots

Each Web Site (within the web server software) has an on-disk folder from which, by default, documents are retrieved using the http protocol. This is the so called **Document Root**.

Virtual roots are used to override the default on-disk location.

Because WebHub applications often serve only a portion of a given Web Site, we recommend that beginners always separate the on-disk location of the document root from the virtual directories.

This separation can easily be achieved by accepting the web server software defaults when installing a new Web Site, and following the instructions in this guide for placing static resources for WebHub applications in a folder within the WebHub application project folder.

Each WebHub application uses 2 virtual roots.

1. for static resources (images,css etc) and
2. for the WebHub runner (runisa.dll etc)

We recommend that the virtual directory for the WebHub runner (generally named **/scripts**) be configured to point to the same physical on-disk location for all Web Sites on a given computer.

Appendix C - Installing the WebHub Connection Layer

The WebHub Connection Layer requires HTTP Server software (on the same machine) in order to run. It is recommended that the HTTP Server software be installed first (if not already installed as part of your operating system).

Licensing

To install the WebHub Connection Layer within your Practice Server, Staging Server, or Production Server environment, you require a licensed version of the software.

A licensed version consists of an installer file (a “setup” program) and an unlock key.

When you purchase a license, HREF Customer Service provides instructions for downloading the installer file as well as your unlock key. The unlock key is a licensee name and a serial number.

The URL for purchasing a license is www.href.com/hrefshop

A license may also be purchased through a WebHub reseller, or may be bundled as part of a WebHub based solution.

Files

As at Jan 2007, the name of the installer file is **WebHub Runtime System v2.080 Setup.exe**

Essentially the WebHub Connection Layer consists of a hub, runner, management utilities and configuration files. The hub file is named hub.exe, the runner is named runisa.dll or runbin.exe, the management utilities are named WebHubAdmin.exe, WebHubView.exe, and ZMAdmin.exe, and the configuration files are named WH*.xml

The hub always runs as an invisible service, and by default, its startup mode is “Manual”.

If you are not sure whether you have the WebHub Connection Layer installed, use the Services applet in Control Panel to look for **WebHub Central**. If this service is installed, then this is a good indication that the rest of the WebHub Connection Layer has already been installed.

Running the Setup Program

The Setup program will install all required files for you. We recommend that you carefully consider three of the folder locations: ZaphodsMap root folder, the runner folder and the sessions folder.

ZaphodsMap root folder

"ZaphodsMap" is the name of the configuration sub-system used by WebHub v2.057+. It is an open-source, cross-platform solution made available by HREF Tools Corp. under a Creative Commons license (see www.ZaphodsMap.com). ZaphodsMap is used by all parts of the WebHub System, including the Hub, runners, all WebHub EXEs, and all the utilities.

The ZaphodsMap system replaces the use of the Windows registry for storing configuration details. It is available for use on Windows and Linux.

The root folder will contain a series of "keybox" files which point to the half-dozen configuration files used by WebHub. These keybox files are very small - only a few k bytes - yet extremely important. It is best to put the root folder in a memorable place that you are sure to back up.

You can move the ZaphodsMap root folder later by changing the global environment variable named ZaphodsMap. (See **Start** > **Settings** > **Control Panel** > **System** > **Advanced**.) If you do that on Windows, you will need to stop+restart the http server to make the new value available to the runner, and you may also need to stop+restart all running WebHub applications and the Hub. For best results, reboot the computer once after changing any global environment variable.

runner folder

The default location for runisa.dll will be c:\inetpub\wwwroot\scripts

and that is a very good location – for most people, in most situations. An alternative location is preferred by people who like to organize their files by software vendor rather than by purpose, and that is

c:\program files\HREFTools\WebHub\bin\whRunner.

In fact, from WebHub's perspective, any folder on any local disk will function equally well. The trick is to remember where you put the runner because you will need to make sure that for each Web Site, **/scripts** (or your chosen scripts virtual directory) points to the folder containing runisa.dll.

You can move the runner later if necessary.

(If you want to use a runner other than runisa.dll, please see the WebHub User's Guide for instructions.)

sessions folder

The default sessions folder will be c:\temp\whSessions. All session variable files (*.var) are “temporary”, in the sense that their contents are valid only for the duration of a site visitor's session. Placing the session files under c:\temp\ is a reminder to the system administrator that the files do not need to be backed up.

A little experience will tell you how much disk space is required for all the session files. A typical range would be from 10kb for a small, low-traffic site to 1mb for a large, high-traffic site.

You can move the session files later if necessary.

Setting Permissions

Regardless of HTTP Server software, the “Local System” account on the computer will need the ability to create, modify and delete files in the sessions folder. This is required because the hub runs as a service under the Local System account, and the Hub needs to be able to control the assignment of session numbers by creating files in the sessions folder.

If you are using Microsoft IIS 5, you need to give Read and Execute rights to the virtual root that points to the runner, runisa.dll. The runner is a “program” because it is a DLL; this is different from running an ASP “script” (which does not require Execute rights).

If you are using Microsoft IIS 6, you have to explicitly enable runisa.dll. Open the IIS manager program, open the "Web Service Extension" node. Click the link "Add a new web service extension". The name can be anything, e.g. "WebHub Runner". Browse to the runisa.dll file. Set the status to "Allowed".

Using an HTTP server other than Microsoft IIS or Apache

WebHub is designed to support many different HTTP servers. If you use an HTTP server other than IIS or Apache (perhaps Sambar from www.sambar.com), then you will need to make some manual adjustments to the WHNetworkInfo.xml configuration file.

An example follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl"
href="http://www.ZaphodsMap.com/zmdesign/css/ZaphodsMap.xsl"?>
<WebHubNetworkInfo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://webhub.com/xsd/cv004/WebHubNetworkInfo200608.xsd"
>
  <Variables begin="{" end="}"/>
  <HttpServers>
    <HttpServer id="Sambar" nameStartsWith="Sambar" sendHttpVersion="true"
sendHttpColon="true" httpForceCase="upper"/>
  </HttpServers>
  <LocalMachine>
    <Runners>
      <Runner id="a" filespec="d:\appsdata\http\scripts\runisa.dll">
        <Interface value="isapi"/>
        <DelayMS value="0"/>
        <UploadFolder value="c:\temp\"/>
        <ErrorLogFolder value="d:\appsdata\WebHub\"/>
        <AdminAccess scheme="http" authority="localhost:80"/>
      </Runner>
    </Runners>
    <ServerProfiles>
      <ServerProfile id="defprofile">
        <HttpServerID value="sambar"/>
        <Scheme value="http"/>
        <Authority value="www.href.com:80"/>
        <RunnerPath value="/scripts/"/>
        <RunnerID value="a"/>
        <URLUseShortFormat value="false"/>
        <URLDelimiter value=":"/>
        <URLForceCase value="lower"/>
        <URLUseLingvo value="false"/>
        <URLUseScheme value="false"/>
      </ServerProfile>
    </ServerProfiles>
  </LocalMachine>
</WebHubNetworkInfo>
```

Defining WebHub AppIDs

You will need to make at least one AppID for your project. You should first decide where your WebHub projects files will be, and then define your AppID accordingly.

For example:

Project root: c:\Clients\MyClient\Project1Name
Desired WebHub AppID: proj1
WebHub Markup Language files:
c:\Clients\Myclient\Project1Name\Live\WebHub\whteko\proj1*.*

If you have CodeGear Delphi installed on the computer, the easiest way to create a new AppID is to use the menu: **Project > WebHub > New AppID**.

To review all AppID definitions, create new ones, and/or edit the properties of an existing AppID, use the **ZMAdmin** utility.

For the above example, you would run ZMAdmin (**Start** | **All Programs** | **HREF Tools** | **WebHub | ZMAdmin**). Under Zaphod Branch Folders, open HREFTools, WebHub, cv004 (meaning configuration version 4). Click on ZMKeybox.xml. On the right panel, under Keys, select "Central Info". Right-click and select **Edit Keyed File**. You will now be editing the WHCentralInfo.xml file for your computer. (Using ZMAdmin provides a standardized way to find the configuration file, which could be anywhere on the computer depending on the installation choices made. Once you know the location of your config files, you do not have to use ZMAdmin to find them.)

Look for the tag <WebHubAppGroups>. Within that can be one or more group definitions, and within each group can be one or more AppID definitions. The definition for the above example would look like this:

```
<WebHubApp appID="proj1">
  <Nickname value="Project1"/>
  <ConfigFolder
      value="c:\Clients\Myclient\Project1Name\Live\WebHub\whteko\proj1\"/>
  <ConfigFilename value="WHAppConfig_proj1.xml"/>
  <ErrorLogFolder value="c:\temp\wherrorlogs\"/>
  <ShareSessions value="true"/>
  <ShareByFile value="true"/>
</WebHubApp>
```

Using the WebHub Utilities

On Practice and Staging servers, it is okay to run WebHubView to observe the hub's work. However, on Production servers, you should only run WebHubView when absolutely necessary, because its presence interferes slightly with the hub's operation and performance may be impacted, especially on servers with two or more CPUs.

Appendix D - Compiling a WebHubApp EXE for use with Dreamweaver

As at version 2.080 of WebHub, the following units are dedicated to interfacing with Dreamweaver 8:

FileName	*Required?*	*Usage*	*Notes*
whdw_RemotePages.pas	yes	use (unaltered)	contains webactions and helper functions that are called from the *remote-design* and *remote-preview* pages
whsample_DWSecurity.pas	yes	copy then modify	requires 4 settings in the XML file that are discussed in Appendix E
whpanel_RemotePages.pas	optional	use, or copy then modify	Panel that displays configuration info and hints and data when remote design requests are processed. It is basically for troubleshooting and education. Feel free to copy and modify and change the GUI as needed
whdw_Comm.pas	yes	used automatically	
whdw_Util.pas	yes	used automatically	

Required additions to the project source (DPR file)

Firstly, backup you project source code.

Open Delphi.

File | **Open** whsample_DWSecurity.pas from the WebHub\lib folder.

File | **Save As** this unit to your project source folder, using a name such as project1_DWSecurity.pas

File |Close All

Then open your project and use **Project** | **View Source** to see your DPR file.

Modify your DPR file in a similar fashion to the following example:

Note 1: in the example, drive h: is assumed to be the location of the WebHub component library (e.g. c:\Program Files\HREF Tools\WebHub\lib). h: can be assigned either by using windows explorer Tools | Map Network Drive to map h: to a pre-existing share, or by using the DOS substitution command from a batch file that runs when you login to windows.
e.g. subst h: "c:\Program Files\HREF Tools\WebHub\lib"

```
program Project1Name;

uses
  SysUtils,
  SvcApp in 'RunAsService\SvcApp.pas',
  SvcObj in 'RunAsService\SvcObj.pas',
  NTObj in 'RunAsService\Ntobj.pas',
  SvcComp in 'RunAsService\SvcComp.pas',
  WebApp,
  utPanFrm in 'h:\utPanFrm.pas' {utParentForm},
  utMainFm in 'h:\utMainFm.pas' {fmMainForm},
  utTrayFm in 'h:\utTrayFm.pas' {fmTrayForm},
  dmWebHub in 'h:\dmWebHub.pas' {dmWebHubCore: TDataModule},
  whMain in 'h:\whMain.pas' {fmWebHubMainForm},
  dmWHApp in 'h:\dmWHApp.pas' {dmWebHubApp: TDataModule},
  whhtml in 'h:\whHtml.pas' {fmAppHTML},
  whappin in 'h:\whAppIn.pas' {fmAppIn},
  whAppOut in 'h:\whAppOut.pas' {fmAppOut},
  whdw_RemotePages in 'h:\whdw_RemotePages.pas'
          {DataModuleDreamWeaver: TDataModule},
  whpanel_RemotePages in 'h:\whpanel_RemotePages.pas'
          {fmWhDreamweaver},
  Project1Name_DWSecurity in 'project1_DWSecurity.pas'
          {dmDWSecurity: TDataModule};

{$R *.RES}

begin
  Application.Initialize;
  Application.CreateForm(TfmWebHubMainForm, fmWebHubMainForm);
  CreateCoreWebHubDataModule;

  pWebApp.AppID := 'demo';

  pWebApp.Refresh;
  pWebApp.Response.HTTPmin := 1;    {http 1.1}
  pWebApp.Security.BuiltInsEnabled := False;

  Application.CreateForm(TfmAppHTML, fmAppHTML);
  Application.CreateForm(TfmAppIn, fmAppIn);
  Application.CreateForm(TfmAppOut, fmAppOut);

  Application.CreateForm(TDataModuleDreamWeaver,
      DataModuleDreamWeaver);
  Application.CreateForm(TfmWhDreamweaver, fmWhDreamweaver);
  Application.CreateForm(TdmDWSecurity, dmDWSecurity);

  InitCoreWebHubDataModule;

  DataModuleDreamWeaver.Init;
  dmDWSecurity.Init;

  Application.Run;
end.
```

Delphi Project Notes

1. The whpanel_RemotePages unit is optional. Use it during development, for troubleshooting and learning. You can safely omit it from the production version of the WebHubApp EXE. The web action components that communicate with Dreamweaver are not in this panel, they are in whdw_RemotePages.
2. Remember to call the Init method on both data modules (as shown, before Application.Run).

Custom changes to security module

The WebHub Programmer should carefully read the code in the project1_DWSecurity.pas file, and decide whether the contained logic is sufficient for the situation or not. It may be necessary to add more rules to increase security or to subtract rules to loosen security.

As an example of an issue that would require customization, the security rules differ depending on whether or not the WebHub application uses a FrontDoor setting. Because the FrontDoor setting forces the first request from a new session to a particular PageID, it effectively blocks a Dreamweaver user from using the remotedesign page whenever the session number is new. The existing sample security module allows for this by providing a bypass for the remotedesign page. If the custom WebHub application already had an OnFrontDoorTriggered event defined, it would be necessary to merge the ideas of the existing event with the ideas in the sample security module – because one can only have a single implementation of OnFrontDoorTriggered.

For free support with security issues, direct questions to the WebHub listserv (sign up at www.href.com/listserv).

Appendix E covers required settings in the WebHubApp's XML file.

Appendix E - Installing and Configuring a WebHub Application EXE for use with Dreamweaver

This Appendix provides an overview of the issues involved in the installation of a WebHub Application EXE, as well as a guide to the configuration steps required for making that EXE function as a dynamic content source for Dreamweaver.

Installation of a WebHubApp EXE

WebHub Application EXEs may be run either as services (more secure) or as applications (more convenient). On a Production Server, security is usually a priority, so the EXE would be installed as a service. On a Practice Server, speed of development is usually a priority, so the EXE usually runs as a normal application. Usage on the Staging Server should be dictated by the usage on the Production Server.

The WebHub Application Developer may provide an installation file which handles most, or all, installation and configuration tasks. The following notes are included in case you want to understand those tasks more fully.

As far as WebHub is concerned, the WebHub Application EXE may be placed in any folder. We recommend placing it into c:\Clients*MyClient**Project1Name*\Live\WebHub\Apps.

"Running the WebHubApp EXE as a Service"

There are two commands used for installing and removing a WebHub Application from Windows services, -install and -remove. They are used from a command prompt as follows:

Project1Name.exe -install
or
Project1Name.exe -remove

Once the EXE has been installed, you can go into the **Control Panel** | **Services applet** to view and adjust its startup mode and parameters. If the EXE requires an AppID or other startup parameters, you must enter those, or it will not function properly. The most common parameter takes this form: /ID=demo The /ID parameter sets the AppID to "demo" when the service starts. Some EXEs have the AppID hardcoded into them and do not require a parameter. If in doubt, check with the WebHub Programmer.

There are two commands used for starting and stopping the service, again used from a command prompt:

```
net start Project1Name
or
net stop Project1Name
```

You can also start and stop the EXE from the **Control Panel** | **Service applet** (assuming you have Administrator or otherwise-sufficient permissions). We recommend that you make some sort of shortcut to stop and start each WebHub Application EXE. Use any technique that you are familiar with, such as a shortcut on the desktop, a .bat file linked from a menu program, a shortcut under your programs area, etc.

Note that some WebHub Application EXEs are invisible when they run as a service, thus it is not obvious whether they are running, or not. You can check by looking in Windows Task Manager, or in the Services applet, or by using the **WebHubView** utility.

"Running the WebHubApp EXE as a Normal Application"

Installing a WebHub Application EXE to run as a normal application is often as simple as copying the EXE from the programmer's computer to a particular folder on the Practice Server.

The EXE is started by double-clicking its icon, by a Windows shortcut, or by running a BAT file which starts it, e.g.

```
start Project1Name.exe
```

If you need to pass parameters to the EXE, be sure to do that in the shortcut or in the BAT file, e.g.

```
start Project1Name.exe /ID=demo
```

Database Aliases, etc.

Some WebHub Application EXEs require a database alias, or database source, to be configured in advance. You will need to check with the WebHub Programmer to find out if there are any such requirements.

WebHub Dreamweaver Interface WHTEKO File

Three WebHub pages are required to create the interface between the WebHub Application EXE and Dreamweaver. Those three pages are loaded from a WHTEKO file (example below). If you do not already have these pages, create a file named **whDWinterface.whteko** and paste the following declarations into them.

```
<!DOCTYPE whteko PUBLIC "-//HREF//DTD whteko stage 2.14//Strict//EN//"
"http://webhub.com/dtd/0214/whteko.dtd">
<whteko
  designdynsrc=""
  defaultlingvo="eng"
  designmode="code"
  designlingvo="eng"
  designpage="Dreamweaver interface pages">

<whpage pageid="remotedesign" desc="For use with Macromedia
Dreamweaver">
<whdoc for="remotedesign">
This page must stay exactly as-is in order for the Dreamweaver
interface to function.
This page implements the Design View feature.
</whdoc>
(~waCommWHDWDLL.execute~)
</whpage>

<whpage pageid="remotepreview" desc="For displaying files in browser
from DW">
<whdoc for="remotepreview">
This page must stay exactly as-is in order for the Dreamweaver
interface to function.
This page implements the Remote Preview feature.
</whdoc>
(~waLoadFileFromCommand.execute~)
</whpage>

<whpage pageid="remoterefresh" desc="For use with Macromedia
Dreamweaver">
<whdoc for="remoterefresh">
This page must stay exactly as-is in order for the Dreamweaver
interface to function.
This page implements the Refresh feature.
</whdoc>
(~app.Refresh~)
</whpage>

</whteko>
```

After saving the interface WHTEKO file, open the WebHub App's XML file and add the new file to the list of files under <TekoFiles>, e.g.

```
<TekoFiles>
<File filename="whDWinterface.whteko" />
<File filename="otherfile.whteko" />
</TekoFiles>
```

To maximize security, you can change the PageIDs of the three pages in the file, so that

strangers can not easily guess the URLs which lead to design features. If you do change the PageIDs, you will need to reference those new PageIDs in the DWRemotePageIDs entry (explained below).

Security Configuration

If the WebHub Developer is using the sample DWSecurity module (very likely), then the following settings must be added to the XML file. Example values are shown and then discussed.

```
<RemoteDesignDefaultInterface>
  <AllowedSessionIDRange value="1430-1440"/>
  <RemoteAccessPages>
     <Page function="refresh" pageID="remoterefresh" />
     <Page function="design" pageID="remotedesign" />
     <Page function="preview" pageID="remotepreview" />
  </RemoteAccessPages >
  <AllowedIPRange value="214.218.44.219;206.201.12.2-206.201.12.62;192.168.1.*"/>
  <TestingServerURLPrefixMappedTo value="c:\temp\DreamweaverPreview\"/>
</RemoteDesignDefaultInterface>
```

The **AllowedSessionIDRange** should encompass the session numbers reserved for use with Dreamweaver. Each Dreamweaver developer should use their own session number, to avoid all potential conflicts. Thus if there are 5 people on the design team, use a range of 5 numbers. You should use numbers between 1001 and 9999. Make sure that each Dreamweaver developer knows their assigned session number, and uses it when configuring their own dyn src within Dreamweaver configuration.

The **RemoteAccessPage** list must list the three pages in the interface whteko file.

The **AllowedIPRange** must list the IP#s used by the design team. As shown in the example, you may list multiple sets of IP#s, and you may use a dash (“-”) to indicate a range of IP numbers.

The **TestingServerURLPrefixMappedTo** entry is required for the Dreamweaver Preview feature. It should be set either to a temporary folder (most convenient) or to the folder containing the WebHub App XML file (for advanced users, single-person-”teams” only).

Load the Interface File and Security Settings

After saving your interface WHTEKO file and making sure that it is included in the <TekoFiles> list, you will need to refresh the WebHub App EXE. This can be done in many ways:

if running as service:

stop and restart the service

if running as application:

right-click on the icon in the tray and select Refresh
or
stop and start the application
or
use the EXE's menu, and select WebHubApp | Refresh

Subsequent Automatic Startup of the EXE

The WebHub Application EXE must be running whenever you want it to serve dynamic content. On a Production Server, that would be 24x7! Thus you should make sure that the EXE starts whenever the server boots.

if running as a service:

Set the service to have a startup mode of “Automatic”
or
Make sure that the “net start *projectname1*.exe” is called whenever the server boots

if running as an application:

Find (or create) the BAT file which runs when the server's main user logs in (i.e. when the server boots), and add a line which starts the EXE, e.g.

```
start projectname1.exe
```

Appendix F Designing a static site with a local HTTP server but without virtual directories

When previewing pages designed in Dreamweaver via a web server, in order to access pages within **separate** website projects, the on-disk location of the **root-folder** for each project must be unique and separate.

Though not recommended, this can be achieved without using virtual directory mappings in the web server

To achieve this (not so good idea), website files would be located in separate project-folders located under the document root of the local HTTP Server software.

e.g. c:\InetPub\wwwroot\Project1Abbrev,
c:\InetPub\wwwroot\Project2Abbrev
etc

e.g. sample Project1 layout

```
c:\InetPub
        \wwwroot                    (document root of local http server)
            \Project1Abbrev         (project-folder & Dreamweaver site root)
                default.html
                contact.html
                \css
                    mystyles.css
                \images
                    image1.png
                    image2.png
                \js
                    myscripts.js
```

Request Example:

resource request	http://localhost/Project1Abbrev/default.html
virtual directory	None
http server root-folder location	c:\InetPub\wwwroot
project1 root-folder location	c:\InetPub\wwwroot**Project1Abbrev**
resource location	c:\InetPub\wwwroot**Project1Abbrev**\default.html

(this assumes that the folder named **Project1Abbrev** has **not** been defined as a virtual directory and that the document root for http://localhost has remained at the default value of c:\InetPub\wwwroot)

The disadvantages with this approach (i.e. not using virtual directories) are:

i) There is no convenient place within this structure for storing documents that relate to the project but are not part of the published website for e.g. preparatory graphic files, specification documents, team coordination documents etc

ii) A more fragmented backup strategy is required

Appendix G: How a browser and http server handle a request for a static web page and resources

Without considering security issues, here is the pseudo English algorithm used by a **web server** for an incoming request URL for a hypothetical static web page named default.html.

Example:

Incoming request URL: http://localhost/Project1Abbrev/default.html

Web Server response:

1. parse the request URL

   host = localhost

   path = /Project1Abbrev/

   resource = `default.html`

2. resolve the host name to an IP number (IP #) {localhost always resolves to 127.0.0.1}

3. if the web server allows more than one domain to exist, determine the "web site" that serves this IP#, else assume the default web site

4. for the determined 'web site', check if the 1st folder in the path (Project1Abbrev) has been defined as a virtual directory

5. examine the file extension (.html) and determine whether any special rules are associated with it (e.g. file extension mapping for .php). in this example, the resource is a static file (default.html) without any special rules, so it will be served directly from disk.

6. as this resource is to be served directly from disk, determine the physical location on disk.

   i) if **no virtual directory** named **Project1Abbrev** exists, then the file will be located in the websites document root e.g. . c:\InetPub\wwwroot\Project1Abbrev\default.html

   ii) if a virtual directory named **Project1Abbrev** exists, then use the directory mapping defined for this virtual directory to locate the file on disk.

   if **Project1Abbrev** maps to

   c:\Clients\MyClient\Project1Name\WebRoot\Project1Abbrev

   then the file location will be

   c:\Clients\MyClient\Project1Name\WebRoot\Project1Abbrev\default.html

7. respond with the contents of the default.html file.

The **browser** would then request each of the resources referenced in the html code of the default.html file.

This is how the browser translates resource references:

1. if a base href tag exists in the html head section, then the browser translates the resource reference to be relative to the url in the base href tag
   e.g. <base href="http://href.com/foldername/" />

2. if no `base href` tag exists, then the browser translates the resource reference dependent on the first character of that reference
   i) for resources starting with "/", references are relative to the HTTP Server document root
   ii) for a reference starting with any other character, references are relative to the location of the currently loaded (html) file.

Appendix H - DOCTYPES that work

DOCTYPES are essential to the proper rendering and functioning of web documents in compliant browsers.

The following is a list of common DOCTYPE tags that work and use valid DTD's as published by the W3C. These are reproduced from more complete listings on http://www.w3.org/QA/2002/04/valid-dtd-list.html

HTML 4.01 - Strict, Transitional, Frameset:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
```

XHTML 1.0 - Strict, Transitional, Frameset:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

XHTML 1.1 - DTD:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
   "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

also see http://www.alistapart.com/articles/doctype/

Appendix I - Advanced Tips

Using a macro to include the <!DOCTYPE tag at the start of each WebHub page.

To avoid the need for explicitly including a <!DOCTYPE tag at the start of each WebHub page in a teko file, it is recommended that a macro be defined in a whteko file that contains shared macros and droplets.

i.e. instead of

```
<whpage pageid="default">
<whoutput>
<![CDATA[<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">]]>
<html>
<head>
  <title>Document title</title>
and so on
```

you would use

```
<whpage pageid="default">
<whoutput>
(~mcDocType~)
<html>
<head>
  <title>Document title</title>
and so on
```

and in the file say `shared.whteko` you would have

```
<whmacros>
mcDocType=(~FLUSH|out~)<![CDATA[<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">]]>
mcSetHeaderForNoCache=(~HEADER|Cache-Control: no-store, no-cache, must-revalidate~)
</whmacros>
```

Thus for each page, `(~mcDocType~)` is expanded to

```
(~FLUSH|out~)<![CDATA[<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">]]>
```

This will flush the output buffer and start the output of your xhtml content with

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Note: `(~mcSetHeaderForNoCache~)` can be called within the <whprep_tag> of any page that you do not want to be cached by the browser.

/* the end */