

# A Web CMS with WebDAV, XML and SOAP

by Chris Harrington



Chris Harrington is a Software Engineer in Pittsburgh, Pennsylvania. Chris graduated from Carnegie Mellon in 1988 with a triple major in Electrical Engineering, Computer Engineering, and Applied Math. He also received a Master's of Science 1993 from the University of Toronto. He began his software career in 1989 writing a "browser" used to provide graphical views of nuclear power plants. He joined DataViews in 1992, where he worked on many commercial and military real time visualizations and command control systems. On a business trip to Vancouver in the early 90s, he visited SFU and saw a room full of students using something he found out was a World Wide Web browser. Immediately hooked, he began working with this new medium and converting DataViews' technology to run within the early web browsers. In 1997 Chris began working as an independent consultant building web applications on the Microsoft "DNA" platform. Chris has built many data driven web sites, including sites which receive over a million page views on busy days. Currently, Chris is employed with FreeMarkets, Inc. where he builds software to automate business processes around market making. He is also an adjunct professor at Carnegie Mellon where he teaches Web Services. Chris volunteers as an active participant in the Pittsburgh SQL Server and XML users groups.

---

## Summary

---

The objective of a Web Content Management System (CMS) is to simplify and streamline the process of creating high-quality, accessible web sites. ThinCMS is a research project which demonstrates the application of current Internet technologies and standards to the task of web site creation and maintenance. ThinCMS is an outgrowth of the authors' Web Services course at the Heinz School of Carnegie Mellon in Pittsburgh, Pa. and of consulting work over the years on web sites and web site creation tools.

The impetus for ThinCMS came from a frustration with the standard industry practices in mapping web design onto implementation, which typically follow an ad-hoc bottom-up creation process using server-side includes (SSI). SSI was an important starting-point, with which the author built several successful web sites. But a recent project was taken as an opportunity to address a nagging feeling that a better approach was possible. The decision was made to abandon the SSI approach and to build a top-down framework of web page composition. This framework is based upon an XML vocabulary called the Web Page Composition Markup Language (WPCML). ThinCMS uses document generators to transform a site's resources into web pages or web scripts. The simple markup structure of WPCMS allowed for the development of a browser-based resource editor, which communicates via SOAP and WebDAV to the workflow engine and resource repositories. The results to date of this project are presented.

---

## A Web Page Composition Markup Language

---

### Frustrations with bottom-up page composition

Page composition is the process of building web pages from logical representations of page structure, and using tools to generate HTML from that structure. For most web site developers, this typically involves looking for repeated design elements, and extracting these out into server-side includes (SSIs). While SSI was a major improvement over the duplication of repeated elements, it is typically pursued in a bottom-up fashion. The page creator will typically build a new page by copying an existing page and then changing the

content. This content page (bottom) includes constructs from higher levels (up) in a logical content/layout inheritance tree.

The problem with this type of bottom-up composition is that one inevitably finds, after creating dozens of pages, that you want to make some small but significant changes to something which has been replicated into each page instance. Since this replication was done at edit/design-time instead of at build/run-time, the site developer will need to edit (by hand or with a script) all of these files to make the change. Granted, there are other approaches for reuse involving parameterized server-side includes. But these usually put one down the road of using a server-side scripting engine and/or web application server to programmatically deliver pages. It is the author's opinion that content creation should be decoupled from content delivery.

### Envisioning top-down composition

There are many high-quality commercial and open-source Web Content Management System (Web CMS) applications available today which aim to streamline web site creation and maintenance. But the author has chosen to pursue his own vision of a light-weight Web CMS, and to focus specifically on creating a framework for top-down page composition using XSLT and other modern web standards and technologies.

Since the decision was made to use XSLT, the process started with a review of existing samples of web site creation using XSLT. Such examples are still scarce. An important finding came from a re-reading of Khun Yee Fung's XSLT book, which gave renewed inspiration and insight. Chapter 12 provides a case study in HTML generation which demonstrated top-down composition. As an exercise, the author recreated this sample web site using an XSLT processor. This exercise demonstrated that the XSLT top-down page composition design pattern was indeed practical.

The seed of a generic multi-level nested template design approach to web development could be seen in the page generator which came out of this exercise. But since all XSLT and XML pages were hand-coded in this exercise, the effort appeared to roll back any advantages gained from this top-down, standards-based approach. What was needed was a more generic representation of web page content and web page composition, and a framework for managing these resources.

### Why WPCML?

Evolving this hand-coded sample site into a generic site generation framework necessitated the emergence of conventions and grammars for the resources which feed the

page generator. These conventions and grammars are what are referred to by WPCML. The need for a framework was largely driven by the goal of building a browser-based editor, which meant ad-hoc, one-off document structures would not suffice. WPCML also represented a recognition that a general-purpose, schema-driven XML editor, while ideal, was beyond the scope and resources of this project.

WPCML uses a reduced and normalized XML structure to represent document composition using a small number of different tags. A dependence upon XHTML and CSS for most content is what allows this simple schema to still represent a very large universe of possible web sites. By changing the problem from one of schema-driven XML editing to WPCML editing, the complexity of the user interface creation task was reduced to one which was manageable.

---

## ThinCMS Framework

---

As with most applications, ThinCMS combines a platform with conventions and application logic to create a framework. The current ThinCMS platform is .NET and Windows 200X Server. The conventions involve the layout of a WebDAV repository and the WPCML schema for XML content. The application logic implements the transformation of resources into pages. In ThinCMS, this application logic is implemented as a SOAP web service. The major components which comprise the ThinCMS framework are described below.

### Page Template Model

ThinCMS models pages as having three levels of nested content/layout components, referred to as "Level" resources. ThinCMS also supports included resources through `iElement` resources, which are content/layout components which can be shared and reused within a site. Element resources can be included in any of the three types of level resources and also into other element resources. Level and Element resource types may be referred to elsewhere as Nested and Referenced resources respectively.

### Level Resources

Level resources are used to define a three-level page structure inheritance tree. The named levels are `iSite`, `iSection`, and `iPage`. It was found in studying many site designs that three levels of nested composition was sufficient to model most sites. And of those which did not, many would probably have benefited from this design constraint.

The Site resource provides the outer composition which is shared by all pages in the site. They will typically contain branding elements (logos, graphics, etc.), site-wide navigation controls (top menu, search form, etc.) and footer text (copyright info, contact info, etc.).

Section resources contain design and content elements which are not global to the site but which are shared by a collection of pages or a section of the site. This level might be used, for example, in an Intranet to allow each department to have its own look. In a public web site, they may be used to give the corporate section a different look from the product section of the web site.

Finally, the Page resources contain content specific to an individual page. Page resources will focus on content, but may also contain Element Parts. Every HTML document in the site instantiates a triplet of templates from the Site level to the Page level. Another way to visualize the page three-level page design scheme is to draw it as a nested template hierarchy. Figure 1 graphically depicts content and template nesting. Referring to this diagram, each content resource on the left gets rendered by the template resource pointed to on the right.

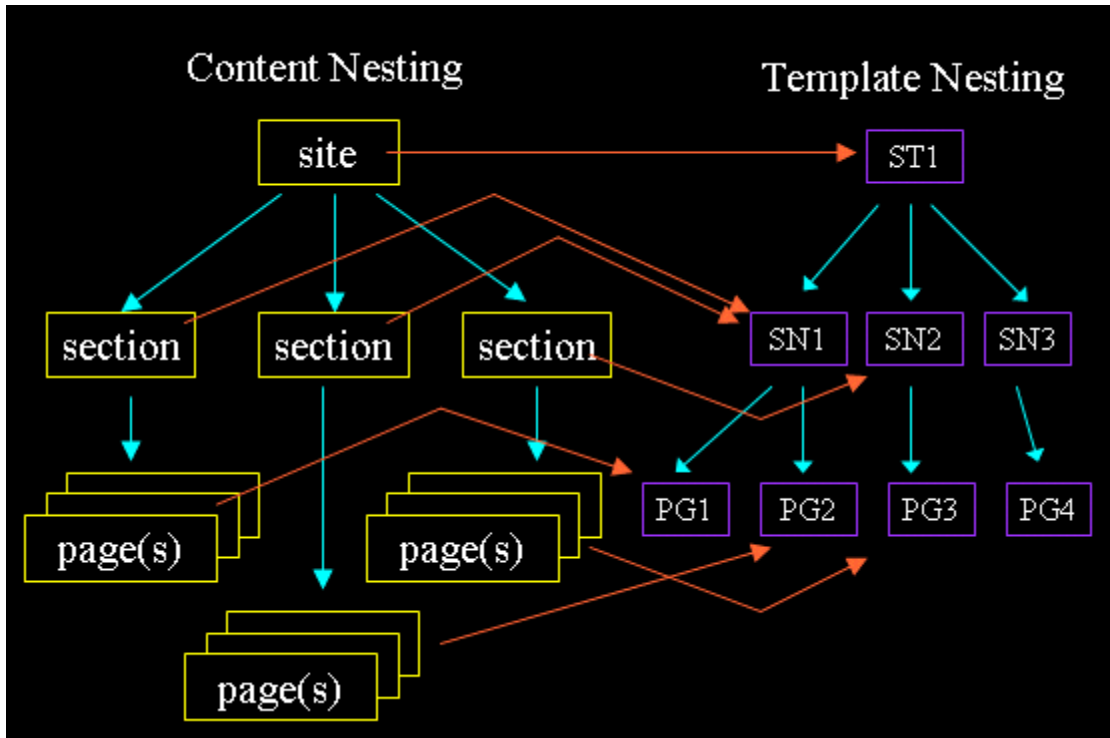


Figure 1. Content and template nesting

### Element Resources

WPCML supports three types of referenced resources. They are Static elements, Navigation elements, and Dynamic elements. A Static element is merely a resource following the WPCML schema which can be included by reference into Site, Layout, or Page level resources (or into other Element resources). Static elements provide the same type of reuse semantics as do server-side includes.

A good example of included content is a vertical news callout placed outside the flow of copy on the right side of selected pages. An included element allows this same content to be inserted into other pages, including pages which have different layouts. Contrast this with level resources. If the news callout was placed in the Section level, then this element would be present only in pages which inherited that layout. And it would be present in all such pages. Element resources provides for a more fine-granular control of content reuse.

Navigation elements are used to represent any navigation controls in pages. Navigation elements receive content from a shared sitemap resource. Many generic XHTML navigation elements have been built in WPCML including breadcrumbs, horizontal and simple menubars, and dynamic hierarchical menus.

Dynamic elements are XML data sources resolved at run-time. They may use static URLs or parameterized (HTTP GET) URLs. Resource names and properties may be used as run-time parameter to this URL. Existing dynamic syndicated content feeds which implement XML web services can be brought into ThinCMS as dynamic elements. SOAP web services using GET semantics can be invoked by Dynamic elements, and the results obtained become content to be transformed and incorporated into the document response. Dynamic elements require that page generation be targeted to a web server which supports a run-time script interpreter with XML and XSLT libraries. This is the case for all current-generation web server platforms such as ASP, ASP.NET, JSP, and PHP.

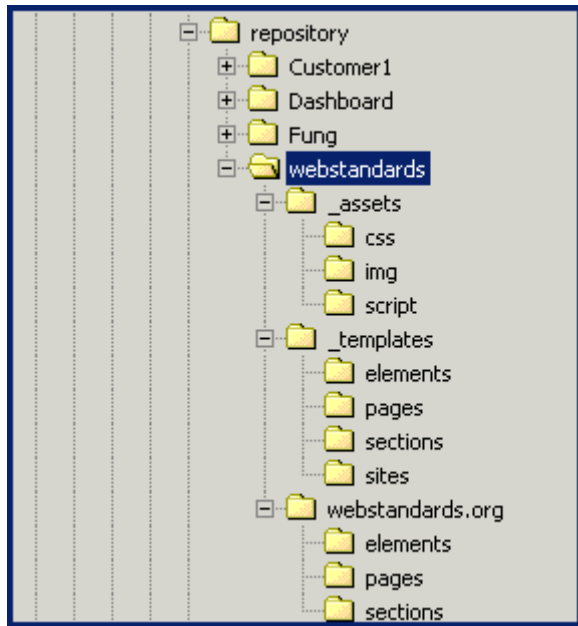
### Resource XML grammar

Each level and element resource follows a simple XML grammar involving nested entity tags. Entities can have properties and content. This simple grammar took inspiration from Joe Sloviski's *iAdvanced UI Design* article series. This simple schema is important as WPCML is the bridge between the UI which manages part resources and the generators which transform part resource into previews and into published pages.

### WebDAV Site Resource Repository

ThinCMS maintains all site resource in one or more WebDAV repositories. Resources are organized into company folders and then into site folders. Each customer

has a templates folder. The templates folder contains folders for each level and element resource template, and an assets folder for resources referenced by the generated HTML documents (images, css, Flash, etc.). Figure 2 below shows a typical resource structure.



**Figure 2.** Typical WebDAV folder structure for site resources

The plan for ThinCMS is to migrate to DeltaV based repositories. DeltaV is an emerging IETF specification for an HTTP-based versioned resource store. DeltaV adds versioning extensions to WebDAV. Using such a server removed the burden of version management from the application developer. Another alternative implementation uses the file system as the resource repository.

## ThinCMS Generators

Three generators implement transformations of resources into viewable content. All three provide "preview" generation of content in the user interface. The page generator is also used to publish the final pages.

### Page Generator

The Page Generator creates HTML pages (or other media types) by applying the ThinCMS transformation pipeline to site resources. The page generator is at the heart of ThinCMS. It is responsible for such tasks as a) generating a composite web page document, b) setting metadata values, and c) delivering generated document to one or more web servers. These three processing steps occur in sequence during each page generation operation.

Composite page generation is accomplished by programmatically generating an outer XSLT template which references all level resources for the composite page and initiates the transformation processor against a dummy document stub. Pages can be generated in preview mode or in publish mode. In preview mode, the links to other managed pages are created as links back to the page generator. Preview generation is performed after edit operations to allow the user to immediately see the results of the edit.

Finally, the page generator has responsibility for sending the resulting document to its intended destination. If invoked in publish mode, the page generator creates run-time versions of href and src HTML attributes. Additional parameters to the publisher tell it how to publish the page (FTP, WebDAV, FrontPage, etc.).

### Element Generator

The Element Generator provides an element preview. This preview allows the user to browse the elements which have been created for a site, before perhaps adding the element into another resource. The element generator uses a special Site Level resource to build the outer HTML container for the element, since the element itself is insufficient to create a complete HTML document.

### Template Generator

The Template Generator provides a template preview. Each level and element content resource refers to an XSLT template which renders the content. These templates can be previewed during site development to allow the user see how content resources rendered with will appear. A test content resource is provided by the designer for every template resource. These test resources are used by the template generator during preview, and are also as content templates by the user interface. The element generator and template generator are only used within the user interface for performing previews.

---

## ThinCMS Workflow

---

The workflow model implemented by ThinCMS is very simple. There are several "times" involved: Design-time, Edit-time, Build-time, and Run-time. Design-time is when the XSLT stylesheets which implement a sites design are chosen or created or modified. This is a task best done by someone skilled in XHTML and XSLT, and a willingness to learn the ThinCMS schemas and conventions. A growing palette of

templates should come into existence as more web projects are built with ThinCMS. These templates are an excellent starting point or learning tool for a designer building a new site.

Edit-time refers to the time one is creating or editing web pages or the sitemap. This task can be accomplished either a) by editing XML files by hand, or b) by using the ThinCMS browser-based user interface. The test content pages which accompany each template are used to "seed" the user interface with objects which can be editor or, in the case of list entities, replicated within the list.

Build-time refers to the time when one instructs ThinCMS to generate pages (and perhaps publish these pages to a web server). Sites can be built using command-line tools on the server, or using the browser-based interface from a remote location. Pages are published to the target(s) specified in a site configuration file.

Run-time refers to the time when a visitor is viewing the generated web site. For pages using Dynamic elements, additional XML content acquisition and transformation can take place at this time.

The idea of clearly separating a site's layouts and styles from its content is one which still does not have universal support within the web development community. For ThinCMS, it was clearly the right decision, both from an architectural and a business perspective. The authors view is that successful web site development requires the intersection of three different skills and capabilities. One is design, one is content, and the third is technology. ThinCMS gives clear responsibility to each of these three roles. The designer and/or information architect has responsibility for design-time template creation. The content owner/author places verbiage and images into these templates, and creates the navigation flow. The web hosting provider maintains a robust, high-performance platform for run-time delivery of static and dynamic content. Applications such as ThinCMS will either be offered by hosting providers or by more specialized application service providers (ASPs).

---

## ThinCMS Application Tiers

---

### Resource catalog

ThinCMS resources are stored in a WebDAV repository which has been described earlier. Because WebDAV follows open standards, any conformant WebDAV server can act as the resource repository tier for ThinCMS.

### Web Service

The application server tier of ThinCMS is a SOAP web service. Application services exposed as XML web services are accessible by any modern hardware/software platform. The application tier implements the Edit-time and Build-time workflow services. The ThinCMS web service tier is implemented in C# on the Microsoft .NET framework. There are no strong dependencies on the .NET platform and so a Java implementation/port could be easily accomplished.

Edit-time services are provided for creating, retrieving and updating the various WPCML resource types. There are about two dozen SOAP methods in this category. These methods are permutations of resource types and resource actions. Resource types include Templates, Parts, Sitemaps, Customers, etc. Resource actions include Add, Update, Delete, Query, etc. For example, the function PageUpdate() is used to update a Page resource instance in the resource catalog.

The application logic determines what published pages are affected by operations performed against the repository using the web service. Effected pages can be published immediately or marked for future scheduled publication. Build-time services are provided for publishing pages to one or more web servers. Page generation is accomplished by invoking the generators previously described.

### User Interface

The ThinCMS presentation tier is implemented as a browser "application". The current implementation is for Internet Explorer. The browser tier makes direct SOAP calls to the web service tier and makes extensive use of XML, XSLT, and DHTML. The user agent tier is written in JavaScript. Since the application tier is a published web service, different user interfaces could be created. The SOAP tier could also act a bridge to other parts of a larger content management system.

The advantage of a SOAP API approach over a page request/response approach for the presentation tier is two-fold. First, the users experience is more like a rich desktop application

because UI components are visually stable (they don't get redrawn on every action due to page reloads). And because all of the presentation tier logic runs on the user's workstation, server load is reduced and interactivity is enhanced.

## Page Generator

The generators can be considered a fourth tier of the application since they operate independently of the application tier and have their own interface semantics, which are URL-based.

---

## The User Interface

---

The ThinCMS user interface strives to limit the number of distracting pop-up windows and dialogs. The interface is implemented in four panels within one resizable window. These panels are described below. Pop-up windows are used for specialized editing tasks, such as HTML content editing and digital assets management.

### Sitemap Panel

While the content panel displays a tree which reflects the layout inheritance of each page, the sitemap panel displays a tree based upon the navigation hierarchy of the pages. This navigation hierarchy is created and managed using the sitetree panel. The target location of a page can be specified by dragging a page from the content tree to the sitemap.

### Resource Panel

This panel contains a number of sub-panels for the different types of WPCML resources. The sub-panels present are Content, Template, Asset, and Element.

### Details Panel

The details panel is also made up of sub-panels. These sub-panels display details about selected resources and entities. The tab buttons on the panel heading are used to select which sub-panel is currently active. Most content editing takes place in detail sub-panels. Whenever a template or content entity is selected in the content panel, its editable details are displayed using textboxes and selection lists. Some values can be entered by dragging objects from the content panels into the detail panels. For instance, the target of a link can be set by dragging the icon of representing the target page into the link value field. Other sub-panels present include a Notes and a Version History.

### Preview Panel

One of the most powerful capabilities of ThinCMS is the site preview. The preview panel occupies the majority of the window real estate. And because the other panel can be collapsed, the preview can be made to occupy almost the whole browser window.

The preview has several purposes. It allows the user to instantly see the results of editing any content entity. It also allows the user to navigate to the particular page which they desire to edit. Finally, the user can select entities within the page, and these entities will become selected in the resource tree which contains the node representing the selected entity.

### WYSIWYG XHTML Editor

Content entities of type HTML can be edited either as HTML markup or using a popup WYSIWYG editing dialog. The formatting capabilities are bare-bones since in XHTML all styles should be applied using stylesheets. The user can choose the HTML tag and the class attribute value from drop-down menus. The current editor does not support in-line images, but this feature and others are on the drawing board.

### Preview-UI synchronization

When the user selects an entity in the preview panel, the corresponding entity is also selected in the content panel. This gives the user a choice of how they select entities to edit. While some users prefer a tree representation, others prefer a rendered page representation. ThinCMS supports both mechanisms. If the template author uses our conventions for style class names, then the entities in the preview will be visually highlighted when selected by the user. The highlight color is used to indicate what type of entity was selected.

---

## Summary

---

ThinCMS and the Web Page Composition Markup Language are evolving into a strong platform for our and our clients needs to rapidly create and then maintain web sites of medium complexity. However, ThinCMS currently does not have the features, like workflow and scheduled publication, to allow it to support larger web sites administered by dozens of people.

By using newer technologies such as XML, XHTML, XSLT, WebDAV, and SOAP, ThinCMS is able to achieve significant sophistication and capability without being a heavy-weight

and complex application. Because all resources are stored as file-based or WebDAV documents, these resources are inherently sharable with other applications. The ThinCMS tiered architecture allows other types of integrations and customizations to be done. For example, a completely different user interface (perhaps non browser-based) could be created which calls the existing SOAP web service tier.