Strategies for Presenting Dynamic Content with Lasso

Peter D Bethke

> "Content is where I expect much of the real money will be made on the Internet, just as it was in broadcasting." -- Bill Gates (1/3/96)
> (http://www.microsoft.com/billgates/columns/1996essay/essay960103.asp).

1. Introduction:

In the continuing evolution of the Internet as an information and entertainment medium, the demand for dynamic, audience-driven content has increased dramatically. Moving from a "static" web site to one that can offer fresh, relevant information takes careful planning and application of Lasso's broad and powerful range of web programming tools.

But what is "Dynamic Content" anyway? There seems to be a general agreement that it is the opposite of "static" content, but from that point the definitions differ. There seems to be a general agreement that Dynamic Content is a more advanced form of web programming than "static" html, but in the modern era of complex CSS this too is arguable.

Obviously, deciding to use Dynamic Content takes a careful analysis of the reasons and necessity, and a careful assessment of the possible drawbacks and/or overhead that may result from it, such as increased server load or search engine compatibility. Some developers address these issues by filtering Dynamic Content to select audiences, also called "personalization" or "authenticated site content". Others focus on optimization, eking out every last bit of performance from their server. Many sites combine the two to create the coveted "how did they do that" reaction.

Dynamic Content can take many forms, from textual to graphic. Some is built "on the fly" and some is delivered pre-built but is considered dynamic due to it's delivery. Using Lasso's powerful PDF and Image tags, it can take the form of generated PDF documents and images. Newer applications use cutting-edge display technologies such as AJAX to push performance of web applications to levels found previously only in desktop applications.

Naturally, all Dynamic Content has to "come from somewhere". Some web applications are based on content that is generated internally, frequently in concert with a back-end database such as MySQL, SQLlite, or MS SQL Server. Others aggregate content that is generated elsewhere, using open standards like SOAP, or using simple included urls or even frame-sets.

Each of these types of applications can be built using technologies that are available to Lasso programmers. Omnipilot has provided developers with a very powerful and flexible array of tools to tackle the issue of Dynamic Content. Using Lasso it is possible to serve Dynamic Content in both textual and binary format (eg images, pdf and other formats) from databases, bypassing the need to store this information in the application directory. Further, this information can be optimized and secured using caching and encryption methods, to create truly flexible and powerful results.

2. Definitions of Dynamic Content on the Web:

A simple search in Google (enter "define:Dynamic Content" in search box) returns a number of definitions of "Dynamic Content". Here are a few that are interesting:

Dynamic Content is:

- ...web site content that can be altered or updated very easily. (www.c7.ca/glossary/)
- Information in web pages which changes automatically, based on database or user information. Search engines will index Dynamic Content in the same way as static content unless the URL includes a ? mark. However, if the URL does include a ? mark, many search engines will ignore the URL. (www.dreamweaverresources.com/seo/glossary.htm)
- Content that is generated on the web pages on the fly. Information gathered from databases or other sources depending on user request, when displayed on the web page makes the content dynamic. Pages that are created in Flash or use other animation technique are also sometimes referred to as dynamic. (www.optymise.co.nz/resources/glossary.asp)
- Content that is assembled to meet users' specific needs, providing them with exactly what they are looking for, when they are looking for it, and in the format they are looking for it in. (www.managingenterprisecontent.com/myweb/Glossary.htm)
- Dynamic Content is the ability to have the presentation of information on a web page, or other services, influenced by other factors. The servers that create the web page run computer programs that, according to a sequence of decisions, alter the content of the page in real-time. Dynamic Content could be as simple as putting the current date in a web page. At its most complex it can identify the person using the page, and personalize the information presented to the preferences they gave to the server when they registered to receive that service. (www.fraw.org.uk/library/005/gn-irt/glossary.html).

If there is a common thread with these definitions it might be that Dynamic Content is content that is presented according to some criteria that is set out by the programmer with some thought for the intended audience, and that this content is changeable based on the provided criteria.

In this case I like the spirit of the last definition, as it it speaks to the role of the developer, who through programming can cause a web server to "...according to a sequence of decisions, alter the content of the page in real-time". This criteria can be as simple as pointing to an included url or using the (date) tag to display the current date to something as complex as presenting content in a specific format at a specific time and place to a specific user at a specific ip range.

Obviously, in the case of the included url or tag, the developer has limited control of the actual content, aside from choosing the source or parameters, or the format of the output. However, the content does meet the criteria of being changeable and presented with some thought towards audience. In the case of the date tag, for example, that thought might be "what is the use of knowing the current time and date". In the case of the highly selective, or "personalized" content, it is easy to make the case that the developer put considerable time into defining the criteria that influenced its presentation.

Finally, the term "Dynamic Content" itself can be confusing as it can refer to the page itself, or elements (objects) on the page. When a definition is referring to the difficulties of optimizing Dynamic Content for search engines, it is primarily talking about the whole page, since search

engines (in general) index the page as a whole. When it talks about date tags or Flash animation it is most of the time talking about objects on a page that might be otherwise static.

3. Advantages/Disadvantages of Dynamic Content

Dynamic Content has obvious advantages. The lure of building a web application that can "run itself" with minimal interaction is great, and is the stuff of many an inticing sales pitch. Bill Gates once famously said (in 1996) that in the modern internet era, "content is king". This is becoming more and more true. Dynamic Content is a vital tool for "visitor retention", whereby visitors to a site are encouraged by what they see at first glance to return later or on a regular basis. Static content is by nature poor at true retention -- while a visitor may return a number of times to read all the content that they need, eventually they will exhaust that "well" and look elsewhere. Even worse, a visitor may "snake" (gather via a web tool) a site for offline browsing, grabbing information, taxing the server, but never actually visiting the site.

The key to visitor retention is to provide information that changes and adapts to time, date, audience, etc, and forces (with a "velvet glove" of course) viewers to return repeatedly to get the "latest", while not appearing limited (or limiting) in any way.

Most times Dynamic Content presented as styled text or a mixture of text and html markup (ie imbedded images). However, advanced techniques of Dynamic Content can go as far as to authenticating and delivering binary content "on the fly" for select audiences. This method, discussed later in this paper, requires a database or an application server that can differentiate between visitor types (or groups). Once implemented, though, the results can be quite powerful, as binary content like images, pdfs, word documents etc can be stored as serialized data and served in a controlled, secure manner that is far more efficient than simply showing or hiding links to "physical" (ie on-disk) files.

The actual implementation of a solid, dynamically-generated application can be very complex. A lot of the complexity comes from the creation of the rules that govern the content itself. Once those rules are created, the code naturally needs to mimic them, and at the same time take into account scenarios that are infrequent and out of the normal scope, such as malformed urls or hacking attempts.

As mentioned in one of the previous definitions, Dynamic Content can pose a problem to indexing services such as Google and Yahoo. Indexing services were first introduced in a time when "static" html pages were the norm, and cgi-based pages were used primarily for simple gateway tasks like email forms. In "modern" times entire applications are built with Dynamic Content -- frequently out of smaller objects that are welded together to create a "rendered" page. This type of page assembly usually requires a scripting or compiled language of some kind, like Lasso, Php, .Net, Java etc. These languages by default use extensions like .lasso, .php etc which tip off the search engine to it's content.

All of these engines deal with Dynamic Content differently, and SEO (Search Engine Optimization) is an art entirely to itself. Of course, truly Dynamic Content is antithetical in a way to the function of the search engine. Because search engines work best with data that remains fixed, Dynamic Content (such as the top news stories on a news site for example) poses problems with search engines that crawl the web periodically - an indexed link might be time-dependent and "vanish" by the time a user accesses it from a search list.

There is also the question of performance, i.e. the load that certain dynamic elements can place on a web server. Many application design decisions are based on a delicate balance between power, complexity and performance. If a web page is made up of multiple Lasso inline "blocks", each representing a separate query, the resulting overhead can potentially slow down server and data source performance, even in a multithreaded environment. A lot of times Dynamic Content can be optimized by spreading a wider net with an initial query and filtering the results using arrays, maps and iterations. Further, using the new cache tags, it is possible to cache the results of complex queries as serialized objects and to call the data when needed, or to refresh and replace it when it is no longer current. In this way the developer can foresee the demands that certain tasks may impose and set up routines that minimize those demands.

The bottom line is that no matter how elegant a site may be, if it is too slow it will have difficulty retaining visitors. This defeats one of the primary reasons for using Dynamic Content, visitor retention, and can make all the hard work of a developer (and client) go to waste.

4. Types of Dynamic Content

There are two major classes of data being served on the web - textual and binary. Textual data usually comes in the form of stylized markup, like HTML or a combination of HTML and CSS. Binary data comes in the form of images, video, PDF, flash animations, etc. Dynamic Content can be any combination of these elements. It can refer to the page that serves the image or the image itself - the definition is largely based on context. Again, what makes it dynamic is that there is some rule or set of rules that governs its appearance, placement and/or accessibility.

Dynamic Content can originate internally, i.e. from a database associated with the application, or externally, from another server or servers. Internally accessed data is the most common form of Dynamic Content used by Lasso developers. It is usually generated by a query to a registered data source, like MySQL, and called by using a named inline or passed to an array either in a raw form or encapsulated into a CType. However the storage and delivery method, at some point it ends up on a display page as text or binary content, subject to the rules governing its appearance and duration.

External referencing and/or presentation of Dynamic Content is sometimes called "Content Aggregation" or "Portalled" content. Some content providers such as Google provide hooks into their own system though SOAP (Simple Object Access Protocol), an XML-based "Messaging Framework". Unlike frames or even externally included urls, SOAP-generated content can (usage guidelines permitting) be made to look like it originates internally, complete with formatting, rules etc (Using the SOAP_DefineTag Lasso ctag introduced in LP8, it is possible to use remote SOAP services as easily as calling local tags).

Javascript can be used in conjunction with Lasso to create Dynamic Content. Javascript has the advantage of widespread browser support, and equally important, browser integration. Browser vendors allow javascript varied levels of access to GUI elements and system information that cannot be duplicated by Lasso. In addition, Javascript is a very capable scripting language that can deliver Dynamic Content by itself or in conjunction with Lasso. Similarly Java, which is more than capable of delivering Dynamic Content, can be integrated with Lasso using the LJAPI (Lasso Java API) suite.

Recently there has been increasing buzz the Lasso community about AJAX (Asynchronous JavaScript and XML). AJAX is not Dynamic Content per se but rather a means to displaying Dynamic Content, much like traditional html-based tag systems but infinitely more efficient. AJAX effectively eliminates or greatly reduces one of the major complaints of Dynamic Content, which is load-time. This can bring web application performance more on par with desktop applications, which itself is the goal of those like Microsoft who see the future as a massive distributed system of centrally leased applications tied together by the internet.

AJAX is an amazing technology that is growing in leaps and bounds, and currently it's only drawback is compatibility with older browsers. However, as with any technology (like CSS for example), AJAX will eventually become very widespread if current trends continue, and it will prove to be a valuable tool in an lasso developer's toolbox.

5. Examples/Scenarios for using Dynamic Content in Lasso

The simplest application of Dynamic Content would be a Lasso substitution tag, such as [date] or [server_date]. When placed on a page, this tag will simply indicate the current date as returned by the server. However, suppose that the developer wanted to vary the date based on the approximate location of the client who was browsing the site. After all, the date and time in Charlottesville, Virginia may be useless to someone in Shanghai, China. It is simple enough to grab the client's ip address using the [client_ip] tag. From this step there are databases available that will match ip addresses to physical locations (humorously referred in internet lingo as "cyberspace to meatspace").

It's not an exact science, though. For example, I recently visited one of these services, ip-to-location.com, which showed that I was located in Florida. A quick check out the window at the snow on the ground showed that I was not. The service had keyed to the dynamically-assigned ip address from my provider (Sprint DSL), which probably originates in Florida. However, it did place me on the east coast in the EST time zone, in the USA. This information could be used in conjunction with the [date_format] to adjust the time-zone offset, based on the client's assumed location and the location of the lasso server. Similarly, one might wish to modify the browser header to specify the language or character set used to deliver content based on physical location. This technique can be very inexact due to a variety of factors but it does show one example of building up from a simple substitution tag to something more complex by mixing various lasso tags. Lasso provides a nice set of tags that are derived from the client header (which are all prefixed by "Client_"), such as [client_type], which can tell you the type of browser that the client may be using. Again, this information can be unreliable but it does provide a starting point for thinking about how information can be selectively delivered based on the needs of the client. Google, for example, makes use of location technology to serve ads and to route visitors to its various language-based portals.

More complex examples involve database interaction. For example, a news-portal site may keep a table of news stories, each record containing columns for title, author, body, etc. Simply displaying these records might be considered Dynamic Content. However, adding display date fields like "start_date" and "end_date" allows the developer to make the display of the content time-based. If the manager of a news portal wished to display a story onto from January 23rd to February 18th, s/he could set the start dates and end dates in the record, and then when running the query compare the value of [date] to these values (formatted of course for the correct data

source). If the current date fell between the start_date and end_date, the story would appear. If not, it would not be returned in the row set and thus not displayed to site visitors. In this way, it is possible for a site to "run itself" in terms of display. The data "front-loading" (i.e. gathering and entering the records) is obviously still a manual form-based or programatically automated task. Combined with the previous method of ip-tracking, it might even be feasible to deliver a combination of time and location-based news stories to site visitors.

Altering site content based on external information (i.e. information gathered from the client browser, ip address etc) is inherently tricky and often unreliable because the information originates outside of the application's sphere of influence. If a site user can be differentiated internally, subject to rules completely in the control of the developer, the reliability of Dynamic Content can be greatly increased. Internal differentiation of site visitors is often referred to as "Authentication". It usually involves a site user logging in, by applying username and password vs a table of approved users. The result of this action is that the application can track the actions and location of the user by matching a url-based session id vs a table of session variables. One of these session variables is frequently a user key (most likely an integer), which can be matched to the user record (frequently a primary key field) at any time to get information such as user name, etc.

Once a developer can reliably confirm the identity of a site visitor, s/he can begin to tailor content based on that visitor's needs. One type of information stored and frequently used to tailor content is "site preferences" or "user profile". Traditionally stored in site cookies, site preferences and profiles are much more reliably stored in a site database and related to a site user via authentication. Preferences or profile information may range from age, sex, etc. to geographical location, areas of interest, and language. Because this information is entered by the user, not gathered from the user's browser connection, it can be considered more reliable (assuming the user is telling the truth). The preference or profile values can then be compared to rules set up by the developer and included in a query. Say for example the news manager wanted to tailor information based on age range as well as date, this could be accomplished using preference and/or profile settings and information if that information had been previously entered by the visitor and stored.

Many developers find a need to assign site users to groups, instead of relying on user-specific information. This has two advantages -- first, it means that a developer does not have to rely on user-provided information to tailor content. It's a fact of web development, users sometimes lie or bend the truth when filling in profiles. Second, once a users/groups system is implemented it is possible to mix and match group assignments to give varying, sometimes overlapping levels of user access. Obviously, the users/groups system is fundamental to operating systems like Unix (and MacOSX of course), and is built into Lasso security. However, it is possible for a Lasso developer to "roll his own" users/groups system to govern site content. Once this is done, it becomes possible to assign group access to Dynamic Content on the query level.

How might this be accomplished? First, lets assume that the developer has two tables, one for users and one for groups. The group table includes the group name, description etc, and of course a primary key which may or may not be the group id. The user table contains first name, last name etc.

Group assignment can be done one of two ways. Either a user can be assigned to a fixed number of groups (for example with a group_id column in the user record which corresponds to the

group id key field in the group table), or a variable number of groups using a related assignment table. Either way it should be possible for the developer to query either the user table or the related group assignment table to determine the groups that a user belongs to. Once this is done these group id's can be queried against a table of Dynamic Content to determine what content can be displayed to the visitor.

Determining what content is associated with what group id can be accomplished in a similar manner. If our developer uses the previously built news story table, with author, title, body etc, s/he can link news items with groups via a group_id key field (locking the content essentially to one group) or a related assignment table that associates group_id with news_item_id. The latter method is much more flexible, as it can make an unlimited amount of associations, which can be also removed without touching the data in either the group or news content tables.

The end result of this is that content queries are generated based on user group, and thus content can be tailored to specific groups of site users without having to rely on browser information. If related tables are used, it becomes possible for developers to create management "back-end" GUI applications to handle the assignment of the various combinations of user, group and content id's.

For example, user "A" might belong to group "A", through an assignment table that associates his user id with the group "A" group id. However, the content that he wants to see is currently associated with Group "B". When he logs in, he cannot view it because the query that governs what is displayed only shows that content that he is authenticated to see, and he is not assigned to a group that can see this data. In order for user "A" to see this content, he must be either 1) added to group "B", or 2) the content that he wishes to see must be associated with group "A". This demonstrates the flexibility that the developer can provide for the site administrator in showing or hiding content. If group "B" was a group that required a monthly fee to belong to, this scenario could easily become the basis for a news site that offered "premium content" to paid subscribers. Non-paid site users might all belong to group "A", and "promoting" a user after payment would be as easy as assigning or associating a new group with his user account/profile.

This can be used with textual output naturally but also with binary content as well. As discussed previously, binary content can be images, pdf files, word documents etc. If binary content is stored as a record in a database table, for example a mysql binary field or longtext field, it can be treated with the same criteria as text output. In other words the same kind of query that matches a text-based "news story" might also match a record that contained a serialized image or pdf file. Using the lasso [image] tags, it is then possible to take the serialized information and serve it to site visitors as an image etc, just as a block of text might be.

The actual tags and code used to serve this image or binary data are more complex than displaying simple text, but the concept is the same. Once Dynamic Content, be it textual or binary, is stored in a database, it can be subject to queries that take into account user group assignments, date ranges, user information, or any number of criteria. Once this is accomplished, a developer can build the overlapping layers of display criteria that can lead to a truly powerful application.

One drawback of serving dynamic, binary information is that the overhead can be very taxing on a server, particularly a database server. It might be able to handle a small amount of users triggering a query which returns a large stream of binary or serialized text, but if a large

amount of simultaneous queries hit a server at one time, the result can be a serious slowdown of performance. This can be helped by query caches (for example the MySQL query cache) if they are supported, but since the query is specific to the current user, its only helps if that user views the binary information multiple times (on a page refresh for example), not the group as a whole, or if the query does not specify the user_id (in which case the query might be cached for a whole group of users).

One recent addition to the arsenal of Lasso Developers is the [cache] tag, which allows developers to cache portions of page output. While this can be done a page level, caching can be problematic if a developer relies on the whole site being dynamic. For example, if a whole page is cached, a page refresh might not reflect other dynamic changes in the site, like ad banners, etc which can occur in "real time" with every page refresh or location change.

One solution is to use the [cache_object] tag to cache large binary objects, and to serve these objects from the cache (possibly stored as a serialize array or map) instead of from the database. The Lasso cache tags also allow developers to compress cached data into byte stream format, and to encrypt it as well. The specific methods for this are beyond the scope of this document, which is intended to be a general overview. However, it is worth using the cache tags, in particular the [cache_object] tag in situations where database overhead is anticipated. Effective use of these tags can significantly improve server speed, which of course is one of the primary requirements of customer retention.

Summary:

In the continuing evolution of the internet, "Content is King". The ability for a web developer to deliver fresh, focused content is critical for those types of applications that rely on customer retention and search engine visibility. There are many strategies for approaching the effective delivery of Dynamic Content, but most require an understanding of the needs of site visitors, and the creation of rules that tailor content to these needs. Lasso provides many tools for delivering Dynamic Content, and it is up to the developer to match the method with the needs of his or her application. If the needs of the application environment, such as performance, user tracking, and display criteria, can be effectively balanced against the needs of the client user, the end result can be extremely powerful.