# Client-Side Dynamic Metadata in Web 2.0

John Stamey, Jean-Louis Lassez, Daniel Boorn, Ryan Rossi
Department of Computer Science, Coastal Carolina University, Conway, SC 29528
{jwstamey, jlassez, deboorn, raross} @ coastal.edu

## ABSTRACT
Web 2.0 brings with it new opportunities for deploying highly interactive webpages. A challenge for Web 2.0 developers is capturing information to document the user experience. Client-side dynamic metadata provides one solution to capturing and logging information on the client-side, while lowering the bandwidth requirements for communication with the server. This paper discusses the architecture of a proof-of-concept system, the Dynamic Metadata Prototype. The architecture is described as an AJAX Design Pattern.

## Categories and Subject Descriptors
D.2.2 **[Software Engineering]:**
Requirements/Specifications - Methodologies

**General Terms:** Design, Documentation

**Keywords:** Metadata, Design Patterns, Dynamic Metadata

## 1. INTRODUCTION
An important characteristic of Web 2.0 is the dramatically increased interactivity users experience through RIAs or *Rich Internet Applications*. [1, 2] Development of RIAs such as Gmail, Flickr.com, and NetVibes.com have been accomplished with AJAX, a suite of technologies originally described by Jesse James Garrett in his essay "Ajax: A New Approach to Web Applications." [3] AJAX is an assemblage of XHTML and CSS, the W3C Document Object Model, XML and XSLT, JavaScript and the XMLHttpRequest Object.

*Classic webpages*, still widely in use, are developed without extensive interactivity. They are created on the server-side and delivered to the browser in HTML/XHTML. While much of the content of classic webpages may be produced by middleware such as PHP, ASP and ColdFusion, there is usually minimal client-side interactivity. [2] Classic webpages are based on a *multi-page interface model,* [4] where new content or changes in content requires delivery of a different webpage to reflect each change. RIAs are delivered with a *single-page interface,* where changes are made to individual components in the webpage, as opposed to refreshing the entire webpage.

Along with increased interactivity, an additional advantage for Web 2.0 applications is better performance. Due to the growing complexity of web presentation technologies, Internet traffic has increased to the point that websites built around the multi-page interface model take longer and longer to load. The heart of this problem is network latency, a measure of how fast a network is running. Network latency is traditionally measured as the time elapsed between sending a Remote Procedure Call to a router and the time it takes for the message to return. The development of Web 2.0 RIAs has worked to help mitigate problems stemming from increased network latency with classic webpages, by reducing data transfer from the server to the client. [5]

A distinct advantage of the Web 2.0 experience is a superior, interactive, user experience. [1] The user experience in the classic multi-page user model is recorded as a sequence of webpages as they are delivered to the browser. [6] Capturing the user experience in the single-page model of Web 2.0 is a more difficult proposition, due to the separation of the client and the server. [7] The alternative to a bandwidth-heavy connection between the client and the server is a true fat client /thin server approach.

This paper discusses an architecture and prototype for a fat client/thin server Web 2.0 application that facilitates logging of the user experience and addresses the problem of network latency. Section 2 discusses server-side metadata, which is currently in use. The Dynamic Metadata Prototype featuring client-side metadata collection is introduced, along with the format used for the metadata. Section 3 introduces the AJAX design pattern, in preparation for its use in describing a reusable solution to implementing client-side dynamic metadata. Section 4 describes the Client-Side Dynamic Metadata Pattern. Section 5 gives the direction for future development with the Pattern.

## 2. METADATA
### *2.1 Server-side Dynamic Metadata*
The idea of tracking user actions has been important since the beginning of the web. [8] However, the bandwidth requirements for constant communication between the client and the server to track and log every user's actions on a webpage can become prohibitive.

Dynamic creation of metadata has been described for a handful of applications; however, all of the applications have been created, used or delivered on the server side. Dognac [9]


*SIGDOC'07*, October 22–24, 2007, El Paso, Texas, USA.

describes the creation of metadata to tag information from ecommerce applications, in lieu of direct database updates. Metadata fields are inserted and updated in a database on the server-side.

Farrel [10] describes a set of components assembled dynamically to create learning objects. Metadata, along with content and contextual information about the learner, is assembled at run-time. Again, the metadata resides on the server, and is updated based on information gathered about the learner during the use of the learning object. Bainbridge [11] describes the configuration and construction of dynamic digital libraries. Metadata is treated as a dynamic part of the digital library insomuch as it can be freely updated as necessary to reflect current information about digital holdings.

Metadata used to hold contextual information is described by Atkas, Fox and Pierce. [12] As information in P2P/Grid Computing environments changes frequently, metadata is an ideal format in which to store information. The P2P/Grid information is stored in a server-side repository. In a related idea, server-side metadata is updated to describe distributed storage systems by Kaul, et. al., implanted in Java with Aspects. [13]

## *2.2 Dynamic Metadata Prototype (DMP)*

The client-side Dynamic Metadata [14] Prototype, DMP, takes approach of logging a user's interactions with a webpage on the client, then storing the information with one connection to the server when the user moves off of a page. This approach is useful with the single-page interface model, because essentially all of a users' activity while they are on a website would be recorded before they move to another website.

DMP is an AJAX-driven website that logs changes in content with client-side dynamic metadata. The current prototype, Version 1.1, is found at www.DynamicMetadata.org. Data from Yahoo weather feed is periodically updated in a DIV. Metadata is created on the client-side to record the changes in content, as well as a timestamp of the change event. The index page of DMP, is seen in *Figure 1*.


*Figure 1*

Several advantages are found in the client-side dynamic metadata prototype system presented herein:

- The processing for all logging is accomplished on the client side, eliminating need to access the server until the very end to store logging information.
- A relatively complete picture of the user can be created by tracking client-side events by taking advantage of client-side processing cycles using Web 2.0 applications.
- User behavior such as mouse movements cannot be accurately relayed to the server without being first stored on the client-side as they occur.

## *2.3 Dynamic Metadata Format*

A two-part format for the dynamic metadata is created on the client-side. Part I uses elements of Dublin Core Metadata, whose specifications may be found at www.DublinCore.org, as a basis from which to start. Dublin Core Metadata has been traditionally used to describe information about learning objects. The online-nature of learning objects lends suitability to a majority of the Dublin Core elements. We present a list of Dublin Core metadata elements that have been modified for use as Part I of dynamic metadata, including any changes necessary for more accurate recording of information:

- Title: name and key of website URL;
- Creator: creator of website URL;
- Publisher: entity responsible for making the resource available;
- Contributor: entity responsible for making contributions to the content of the URL;
- Date: creation date or date of last update of URL;
- Format: resource platform version of URL and server, if delivery comes from multiple load-balanced servers (optional);
- Identifier: session variable or unique identifier of user;
- Source: reference to a source providing the content, in whole or in part; if the material has been derived from more than one contributing source, there will be more than one source element;
- Language: native language of the content, in two or three letter, such as EN or ENG for English;
- Coverage: location of resource, and/or dates for which the content is valid; and,
- Rights: rights held by the creator such as a copyright.

The Document Type Definition for Part II of our client-side metadata is found below:

- Sequence_No: sequential order of an event
- DateTime: timestamp of an event, split into Date and Time
- Contents: information that has changed or movements recorded, in the form of a name-value pair – Element_ID, the DIV receiving new information; and, Element_value, the actual value or movement to be recorded

## *2.4 Sample Dynamic Metadata*

Executing the DMP produced the following metadata. In *Figure 2*, we see the metadata created when the URL was first visited. In *Figure 3*, five minutes later, additional information was appended to the metadata in *Figure 2* as a result of an update to DIV with id= weatherDiv.

# 3. PATTERNS

## *3.1 Alexander Patterns*

Design patterns originated with Christopher Alexander [15, 16] in 1977 as a way to describe fundamental building blocks of towns, buildings and construction. Alexander's classic pattern has five parts: [16, pp. x-xi]

```
<metadata xmlns="http://example.org/myapp/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemalocation="http://example.org/myapp/schema.xsd"
xmlns:dc="http://purl.org/dc/elements/1.1/">

<dc:title>www.DynamicMetadata.com</dc:title>
<dc:creator>Stamey, Lassez, Boorn and Rossi</dc:creator>
<dc:publisher>Herald Hosting, Inc.</dc:publisher>
<dc:contributor>Boorn</dc:contributor>
<dc:date>May 1, 2007</dc:date>
<dc:format>LAMP (Linux, Apache, MySQL, PHP)</dc:format>

<dc:identifier>b91442d8d20bfafa2d6f92a8578bd8e9</dc:identifier>
<dc:source>www.yahoo.com, www.lipsum.com</dc:source>
<dc:language>EN</dc:language>
<dc:coverage>No expiration</dc:coverage>
<dc:rights> (c) 2006-2007, Stamey, Lassez, Boorn</dc:rights>

<dm:info>
<dm:sequence_no>1</dm:sequence_no>
<dm:datetime>
   <date>2007-05-30</date>
   <time>1180556912</time>
</dm:datetime>
<dm:contents>
   <dm:element_id>weatherDiv</dm:element_id>
   <dm:element_value>
        <rssweather city="El Paso" condition="Fair"
        temp="87" units="F"
        wspubdate="Wed, 30 May 2007 12:51 pm MDT">
        </rssweather>
    </dm:element_value>
</dm:contents>
</dm:info>
```

*Figure 2*

```
<dm:info>
<dm:sequence_no>2</dm:sequence_no>
<dm:datetime>
  <date>2007-05-30</date>
  <time>1180556917</time>
</dm:datetime>
<dm:contents>
  <dm:element_id>weatherDiv</dm:element_id>
  <dm:element_value>
     <rssweather city="El Paso" condition="Sunny"
      temp="88" units="F"
     wspubdate="Wed, 30 May 2007 12:56 pm MDT">
     </rssweather>
  </dm:element_value>
</dm:contents>
</dm:info>
```

*Figure 3*

1. A picture to show an architectural example of the pattern;
2. An introductory paragraph to set the context of the pattern – giving the essence of the problem in one or two sentences;
3. The body of the problem, describing the empirical background of the pattern, evidence of its validity, and the manner in which the pattern can physically manifest;
4. A set of instructions explaining the solution to the problem, along with a diagram showing the main components of the solution; and,
5. Identification of related patterns that form a (usually small) pattern language.

## *3.2 Gang of Four Design Patterns*

Gamma, Helm, Vlissides and Johnson, the Gang of Four [17] introduced the notion of design patterns in object-oriented programming. Their platform was C++ and Smalltalk, and their diagramming technique was an early version of UML (referred to as the Object Modeling Technique, or OMT). Design Patterns were used to describe reusable solutions in object-oriented programming. Design Patterns were expanded to contain thirteen descriptive sections, the additional granularity needed when the context changed from architecture (Alexander) to Object-Oriented Programming (Gang of Four).

1. Pattern Name and Classification as creational, structural or behavioral;
2. Intent -- what does the pattern do and what problem is addressed;
3. Also Known As -- other well-known names of the pattern;
4. Motivation -- a scenario illustrating a design problem and how the class and object structure of this particular pattern solve the problem;
5. Applicability -- situations in which the design pattern can be applied;
6. Structure -- an OMT diagram of the pattern;
7. Participants -- a list of the classes and objects embodied in the Structure;
8. Collaboration -- a scheme of how the Participants work together in the solution;
9. Consequences -- potential trade-offs and results of using the pattern;
10. Implementation -- issues that would arise in the implementation of the patter, along with potential language-specific issues;
11. Sample code -- code fragments of the solution;
12. Known Uses -- at least two examples, from different domains, of the pattern used in real-world systems; and,
13. Related Patterns -- other patterns that are frequently used in conjunction with the pattern being explained.

## *3.3 AJAX Design Patterns*

Expanded design needs for Web 2.0 applications have necessitated a further expansion of design patterns used with AJAX [18, 19] Mahemoff's AJAX Design Pattern contains most of the elements of both Alexander and Gang of Four patterns, as well as additional information to guide the implementation.

1. Evidence – On a zero-to-three scale, the author of the pattern is asked to propose the pattern is speculative (0), in existence as a proof-of-concept (1), exists in several examples (2), or claim the pattern is in widespread usage. The claim of evidence in AJAX patterns actually allows the creation and development of patterns without Alexander's criteria of widespread existence.

The actual number assigned to the pattern is based on how many of three small round circles are filled-in.

2. Tags – Tags, or keywords, help convey a sense of focus for the pattern. The Gang of Four's *Also Known As…* section, describing other names for a pattern, is a similar construct.

3. In a Blink – Reminiscent of Alexander's picture at the beginning of his pattern, AJAX patterns begin with a hand-drawn sketch or very brief sentence fragment describing the intent of the pattern. The informality of the hand-drawn sketch and/or sentence fragment is also like the Gang of Four's *Intent*, a short statement about the purpose and intent of the pattern.

4. Goal Story – AJAX patterns relate an actual success story that will be made possible once a RIA, featuring an AJAX pattern has been implemented. The success stories use personae drawn from a cast of nine "typical characters" found in the information technology world, ranging from end users to senior application development programmers. An early and successful use of the personae technique to explain problems and solutions in highly quantitative fields is found in S.J. Simon's classic book on contract bridge strategy, "Why You Lose at Bridge." [20] The idea of getting to know certain personae and their behavioral characteristics to help with understanding quantitative concepts has found its way into the practice of programming. [21] Personae have been used to help explain concepts in the area of software engineering pedagogy. [22, 23]

5. Forces – Forces identify appropriate advantages of RIAs over classic webpages. These include increased application speed (overcoming network latency), increased usability, and the availability of technology for implementation. Forces can be thought of as The Gang of Four's *Consequences*, that discuss how the pattern supports its objectives.

6. Solution – The solution in an AJAX pattern is a one to two sentence summary followed by a general elaboration, including implementation details, rationale, programming aspects, caveats, and potential problems found in the implementation. This scheme is much like Alexander's approach to presenting the solution solved by a design problem. It also embodies the *Structure*, *Participants*, and *Collaborations* from the Gang of Four's pattern.

7. Decisions – Decisions are posed as questions. They describe problems and ideas for their resolution during implementation. Decisions have elements of the Gang of Four's *Implementation*.

8. Real-World Examples - This section gives real-world examples of the pattern at work. For instances where real-world examples are lacking, a proof-of-concept is appropriate. This section is very much like the Gang of Four's *Known Uses*, as well as the body of Alexander's "solution" section in his pattern.

9. Code Example – AJAX patterns require code, the same as the Gang of Four's *Sample Code* and Alexander's diagram requirement in his "solutions" section.

10. Refactoring Illustration – In the spirit of Martin Fowler, who originally defined refactoring as a change in code that would not lead to any user-observable behavior change, [24] this section describes the effect of the pattern to provide traditional refactoring efficiencies on the application.

11. Related Patterns – In this section, AJAX patterns converge perfectly with the Gang of Four and Christopher Alexander.

12. Metaphor – Metaphors are useful to remember patterns. Metaphors blend in with the Gang of Four's *Also Known As…*, which was also associated with AJAX pattern Tags.

13. "Want to Learn More?" – This section contain links to any original references and other relevant material. The purpose for hyperlinks stems from the *www.Ajaxpatterns.org* website, a wiki that is constantly updated with new AJAX patterns and resources.

14. Acknowledgements – Attribution is given to individuals who contributed to the discovery and/or development of the pattern.

## 4. THE CLIENT-SIDE DYNAMIC METADATA DESIGN PATTERN

We now explain the generation of Dynamic Metadata with an AJAX Design Pattern.

1. Evidence –

2. Tags – Metadata, Dynamic Metadata, Client-side metadata

3. In a Blink – Generate a log to track user behaviors in a browser in a well-defined metadata format on the client-side

4. Goal Story – Mamehoff's character, Stuart, is a student with lots of time to listen to music and other hobbies not related in any way to his studies. While Stuart has his web browser open, a number of items on www.dynamicmetadata.com change. New management at Dynamic Metadata want to know what Stuart actually sees in his browser as well as his navigation patterns throughout the website, without having the costly overhead of logging everything server-side. Dynamic Metadata is generated on Stuart's computer to track his activities as well as the changes he experiences in his webpage. When Stuart leaves the website, the tracking data is sent to the server for archival and data analysis.

5. Forces:

- Dynamic Metadata has been developed to provide a standard for sending logging data to the server arising from Web 2.0 sites developed with AJAX. Working within a standard format should lead to uniformity of data throughout the archive.
- Data in DIV elements will be changed as a result of values being placed in the innerHTML property of a DIV. A rapid succession of DIV changes will be handled with the XMLHttpRequest Object that can easily write the metadata. User browsing patterns (mouse clicks and other activities) will also be captured.

- A full page-refresh for each DIV change would consume server resources. Thousands of users could crash a server with rapid-fire page refreshes.

6. Solution – Information encoded in the Dynamic Metadata format will be appended to a text string for each change of content and for each recordable browsing action. This text string will be sent to the server using the JavaScript OnUnload event handler when the user leaves the webpage. A diagram of the program flow of the solution may be seen in *Figure 4*.

The DMP system consists of four files:

- index.html – loads the weather information from Yahoo using the fetch_weather() function, and uploads the accumulated logging metadata to the server with the onUnload property..
- fetchpage.php – forwards XML weather data from Yahoo for parsing.
- logdm.php - saves metadata to the server.
- metadata.js - has AJAX code, XML parsing code, and concatenates each additional dynamic metadata entries into a string.

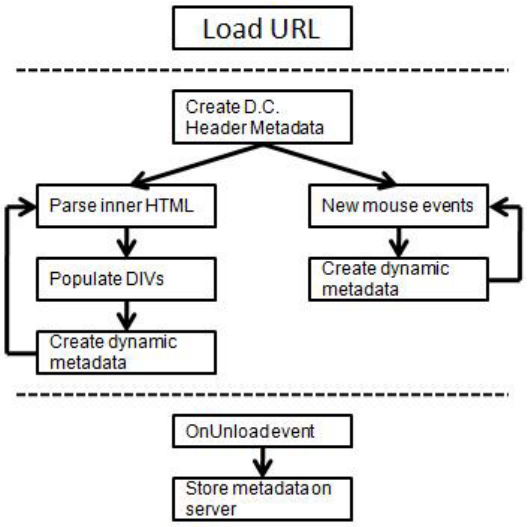


*Figure 4*

7. Decisions
*How can metadata be accumulated on the client-side?* One strategy is by simple concatenation into a text string to be returned to the server. A different strategy involves creation of an XML object that can be updated in a straightforward manner. While either would work, the implementation chosen was simple string concatenation.

*What are disadvantages of client-side dynamic metadata?* Every time the user refreshes an AJAX-driven RIA, the server will receive information as if there was more than one session. However, identifying the session with a cookie can solve this problem. The other problem is when the computer on the client side is shut down before they leave the page. There is no known solution for this problem, unless some mechanism can be set up so that when the computer is rebooted and connected to the Internet that the server is notified of a session that was lost.

8. Real-World Examples – A proof-of-concept has been developed and may be found at http://www.DynamicMetadata.org.

9. Code Example – An important part of this implementation is the use of the Yahoo namespace in the RSS weather feed. Appendix 1 shows how this was accomplished in the actual code from the metadata.js file. The importance of using Yahoo's namespace is that if their data changed, the data on the website could change automatically.

10. Refactoring Illustration – AJAX is important to the efficiencies to be achieved in this dynamic Web 2.0 application. More specifically:

- Foundational AJAX technologies are exhibited with DIVs changing from the use of the innerHTML property, as well as the dynamic metadata being sent to the server OnUnload.
- Many of the DIVs will be receiving data via XML. This will be parsed on the client-side, therefore the server will have no knowledge of the content.
- The only way that content can be logged is on the client-side through generation of logging data.

11. Related Patterns: AJAX App, Web Remoting

12. Metaphor: Client-side logging

13. "Want to Learn More?"
www.Ajaxpatterns.org
www.DynamicMetadata.org

14. Acknowledgements -- Lassez and Stamey (dynamic metadata concept), Stamey (system design), Boorn (code), Rossi (pattern analysis and sequence analysis)

## 5. FURTHER WORK

We describe a first step toward capturing and documenting the user experience in Web 2.0 applications by taking advantages of the AJAX architecture of Rich Internet Applications. The system architecture we propose uses client-side, dynamically generated metadata to log content changes and user actions. A continual connection to the server is avoided, thus decreasing the amount of bandwidth needed to capture this information. The DMP system currently captures changes in DIV tags. A second version, 2.0, will also record mouse movements and clicks, thus providing moredetailed data about user sessions in a single-page interface model.

## 6. REFERENCES

[1] O'Reilly, T. "What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software," Sept. 30, 2005. Retrieved May 1, 2007 from http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html/.

[2] Lin, K-J Building Web 2.0. *IEEE Computer*, May 2007, pp. 101-102.

[3] Garrett, J.J., (2005) "Ajax: A New Approach to Web Applications," retrieved May 1, 2007 from http://www.adaptivepath.com/publications/essays/archives/000385.php/.

[4] Mesbah, A. & van Derusen, An Architectural Style for AJAX. *The Working IEEE/IFIP Conference on Software Architecture (WICSA'07).*

[5] Stamey, J.W. & Richardson, T. Middleware development with AJAX. *Journal of Computing Sciences in Colleges*, Volume 22, Issue 2 (December 2006) Pages: 281 - 287.

[6] Ho, S.Y., An Exploratory Study of Using a User Remote Tracker to Examine Web Users' Personality Traits. *Proceedings of ACM ICEC 05*.

[7] Lewandowski, S. Frameworks for component-based client/server computing. *ACM Computing Surveys*, March 1998.

[8] Srivastava, J., Cooley, R., Deshpande, R. & Tan, P-N. (2000) Web usage mining: discovery and application of usage patterns from Web data. *Proceedings of ACM SIGKDD 2000*, pp. 12=23.

[9] Dogac, A. et al., A Workflow-based Electronic Marketplace on the Web. *Proceedings of SIGMOD*, Vol. 27, No 4. December 1998, pages: 24-31.

[10] Farrell, R., Liburd, S. D., Thomas, J.C., Dynamic Assembly of Learning Objects. *Proceedings of WWW 2004*, ACM Press, pages: 162-169.

[11] Bainbridge, D. et al., Dynamic digital library construction and configuration, *Proceedings of The European conference on Digital Libraries. 2004*.

[12] Aktas, M.S., Fox, G., Pierce, M., Managing Dynamic Metadata as Context. *Proceedings of 2005 Istanbul International Computational Science and Engineering Conference.*

[13] Kaul, D., Gokhale, A., Dawson, L., Tackett, A., & McCauley, K. Applying Aspect Oriented Programming to Distributed Storage Metadata Management. *Proceedings of ACM BPOAOSD 2007*.

[14] Lassez, J-L, Stamey, J.W., Patterns in the Digital Bibliotheque City. Invited Presentation, Digital Bibliotheque City Workshop, Hokkaido University, Sapporo, JP. March, 2006.

[15] Alexander, C. *The Timeless Way of Building*. Oxford University Press, 1977.

[16] Alexander, C. *A Pattern Language.* Oxford University Press, 1977.

[17] Gamma, E., Helm, R., Johnson, R., & Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional, 1995.

[18] Mamehoff, M. (2006) *AJAX Design Patterns*. Sebastapol, CA: O'Reilly Media, Inc.

[19] Mamehoff, M. (2006-7) Pattern Template. Retrieved May 31, 2007 from http://Ajaxpatterns.org/Pattern_Template/.

[20] Simon, S.J. (1946) *Why You Lose at Bridge*. New York: Simon & Schuster.

[21] Rosenberg, A.N. (2001) A Description of One Programmer's Programming Style Revisited. Retrieved May 31, 2007 from http://the-adam.dyndns.org:2069/adam/rantrave/st02.pdf/.

[22] Moritz, S.H., Wei, F., Parvez, S.M. & Blank. G.D. (2005) From Objects-First to Design-First with Multimedia and Intelligent Tutoring. *Proceedings of ACM ITiCSE 2005*.

[23] Pollice, G. (Feb. 15, 2006) Software Development 101. *The Rational Edge*, February 2006. Retrieved May 31, 2007 from http://www.ibm.com/developerworks/rational/library/feb06/pollice/index.html/.

[24] Fowler, M. *Refactoring: Improving the Design of Existing Code.* Addison-Wesley Professional, 1999.

# APPENDIX 1

## Code for Parsing Yahoo Namespace and Building Dynamic Metadata

```
/* Parse Weather Module (xml document object) */
function parseWeather(xmlDoc){
 //Fetch Yahoo Weather namespace URI
 var weatherNsUri = xmlDoc.getAttribute('xmlns:yweather');
 //Create weather node to store weather information from Yahoo's namespace
 var weatherNode = new Object();
 //Fetch user tracking data
 var userid = xmlDoc.getAttribute('userid');
 var sdate = xmlDoc.getAttribute('date');
 var stime = xmlDoc.getAttribute('time');
 //Fetch weather data
 weatherNode.city =
   xmlDoc.getElementsByTagNameNS(weatherNsUri,'location')[0].getAttribute('city'); //City
   Name (e.g. Myrtle Beaach)
 weatherNode.condition =
   xmlDoc.getElementsByTagNameNS(weatherNsUri,'condition')[0].getAttribute('text');
   //Current condition (e.g. sunny)
 weatherNode.temp = xmlDoc.getElementsByTagNameNS(weatherNsUri,
   'condition')[0].getAttribute('temp'); // temperature number (e.g. 88)
 weatherNode.units = xmlDoc.getElementsByTagNameNS(weatherNsUri,
   'units')[0].getAttribute('temperature'); // temperature units (e.g. F)
 weatherNode.wsPubDate = xmlDoc.getElementsByTagNameNS(weatherNsUri,
   'condition')[0].getAttribute('date'); // weather service publish date time
 weatherNode.rssTitle =
   xmlDoc.getElementsByTagName('item')[0].getElementsByTagName('title')[0].firstChild.nodeVa
   lue;
 weatherNode.rssDescription =
   xmlDoc.getElementsByTagName('item')[0].getElementsByTagName('description')[0].firstChild.
   nodeValue;

 //Update weather div
 document.getElementById('weatherDiv').innerHTML = "<div
   class='weatherTitle'>"+weatherNode.rssTitle+"</h2>";
 document.getElementById('weatherDiv').innerHTML += "<div
   class='weatherDescription'>"+weatherNode.rssDescription+"</div>";
ode.rssDescription+"</div>";

 //Build dynamic metadata
 var str = "<dm:info>\n";
 str +=    "  <dm:session_id>"+userid+"</dm:session_id>\n";
 str +=    "  <dm:sequence_no>"+sequenceNum+"</dm:sequence_no>\n";
 str +=    "  <dm:datetime><date>"+sdate+"</date><time>"+stime+"</time></dm:datetime>\n";
 str +=    "<dm:contents>\n";
 str +=    "  <dm:element_id>weatherDiv</dm:element_id>\n";
 str +=    "  <dm:element_value>\n";
 str +=    "  <rssweather city='"+weatherNode.city+"' condition='"+weatherNode.condition+"'
   temp='"+weatherNode.temp+"'

units='"+weatherNode.units+"' wspubdate='"+weatherNode.wsPubDate+"'/>\n";
 str +=    "  </dm:element_value>\n";
 str +=    "</dm:contents>\n";
 str +=    "</dm:info>\n";
```