



REALTIME
FRAMEWORK

Realtime® Framework

Product Positioning and Technology White Paper

January 04, 2012

Abstract

Save up to 90% on server resources and bandwidth.

Realtime® allows the use of real time communication between the users' browsers and the servers, creating an unprecedented amount of interactivity with the website visitor, while greatly reducing bandwidth usage and server load.

The Realtime® Framework is a set of tools and technologies that facilitate the development and deployment of websites using real time. The benefits of using the Realtime® Framework include a high-performance, high-scalability server solution, easy and fast development of products and applications using real time, the ability to use the technology on several platforms and reduced bandwidth usage.

In this white paper we explain why the real time web is important, why the Realtime® Framework is an ideal choice for the development of real time - enabled websites and web applications and provide technical details of the platform's Open Realtime Connectivity (ORTC) and the Extensible RealTime® Markup Language (xRTML).

Sérgio Costa, VP for xRTML

Audience

This document is intended for two types of readers.

The first is the information technology decision maker who wants to learn more about the benefits of developing real time – enabled websites and web applications using the Realtime® Framework. This reader may find the technical details further ahead on this document.

The second is the more technical reader who wants details about how the Realtime® Framework actually works. This reader should find it useful to read the whole document.

Executive Summary

Realtime® allows the use of real time communication between the users' browsers and the servers, creating an unprecedented amount of interactivity with the website visitor, in a cost-effective way, with a low time-to-market time.

The Realtime® Framework is a set of tools and technologies that facilitates the use, development and deployment of real time in web applications, being composed by two layers:

1. The development layer, comprised of the Extensible Realtime® Markup Language (xRTML), which is very similar to HTML, making it easy to learn and use. It also works as an abstraction layer to ORTC, described below. xRTML offers an array of ready-to-use tags (similar to HTML tags) that act as message managers and processors. This gives xRTML the ability to work out-of-the-box, with very little development. Being extensible it also allows developers to further expand the features of already existing tags or make new ones from scratch, taking advantage of the core of the platform and its features.
2. The server and communication layer, through Open Realtime® Connectivity (ORTC), which the developer may actually never come in contact with if he/she decides to use only xRTML for the handling of communications. ORTC provides a series of production proven advantages such as high performance, high scalability and security over other real time solutions in the market.

By using the Realtime® Framework, developers are provided with a way to add the benefit of real time to their web applications in a quick way, without the need to worry about how communications are handled and processed, though they are able to fully control message flow and processing if required. This offers an extraordinary amount of control and power to the programmers, allowing them to develop any kind of real time-enabled web application.

“Not since I was involved in building the business model for YAHOO back in 1995 have I seen such an innovative and disruptive opportunity.” — Mr. Andy Batkin, CEO at Social TV Summit

Legal

The information in this document represents the current view of IBT, S.A. on the issues discussed as of the date of publication. It should not be interpreted to be a commitment on the part of IBT and IBT cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. IBT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Without limiting fair use rights, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of IBT, S.A.

IBT may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in a written license agreement from IBT, this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2011 IBT S.A.. All rights reserved.

Realtime®, xRTML® and ORTC® are either registered trademarks or trademarks of IBT S.A. in Portugal and/or other countries. The names of other companies and products mentioned may be trademarks of their respective owners.

Table of Contents

1. The Real Time Web	1
Why do we Need Real Time Web	1
How Can the Realtime [®] Framework Help You	1
2. How Real Time Works	3
The Request/Response Model.....	3
The Near-Real Time Web.....	4
The Realtime [®] Web	5
A Realtime [®] Case Scenario.....	6
3. The Realtime[®] Framework	8
Open Realtime Connectivity (ORTC)	8
<i>Layer of Abstraction</i>	8
<i>Multiplexing</i>	9
<i>Security</i>	9
<i>Scalability</i>	10
Extensible Realtime [®] Markup Language (xRTML)	11
<i>An Example</i>	12
<i>Cross-Browser and Cross Platform</i>	12
4. Further Reading	13
Realtime.co.....	13
xRTML.org.....	13
What Realtime Means to the Future of the Web	13

The Real Time Web

Why do we Need Real Time Web

Since its beginning, the World Wide Web has greatly evolved. New technologies have emerged; e-commerce is a reality and users are becoming more and more demanding. People want to get their information fast and in a reliable way. They need to know that their data will be displayed to them when they need it. They depend on it, making it the developers' job to provide that. Of course, there are ways to simulate real time and AJAX, for example, is one of them that has served us well for the past few years, but inherently has its downsides and limitations, mostly because of the model over which the web works. Websites are also always looking for better ways to display content and the ability to truly push it to their users. News websites, for example, could use the ability to alert users for breaking news or indicate them whenever a new item is added to their articles section.

Users demand a better service and websites crave to be able to provide them. Whoever is able to deliver to them is more likely to capture their loyalty. Companies succeeding in providing this better service will, as usual, be the ones to win over their competition. Real time web, and the Realtime[®] Framework, in particular is the right tool for the requirements of these new challenges, since it allows companies to stay ahead of the competition, while saving money and cutting costs at the same time.

How Can the Realtime[®] Framework Help You

The Realtime[®] Framework was created from the start to be a valuable asset for developers to create real time websites or to adapt it to their current projects. It provides an all-around, out-of-the-box solution with both real time server technology and an easy to learn, easy to use markup language, very similar to HTML. Programmers using the Realtime[®] Framework will be able to develop and deploy high

performance, highly scalable real time web applications in an orderly development environment, thanks to the abstraction layer it provides, and with a very low time-to-market.

How Real Time Works

The Request/Response Model

The current web model works on a Request/Response model, working over HTTP. The users' browsers contact the web servers, sending a request for content. The web server then replies to the browser, sending the information required by the user (mostly often in the form of HTML pages). Browsers fully control this model, where the server is only able to send information to visitors whenever they ask for content.

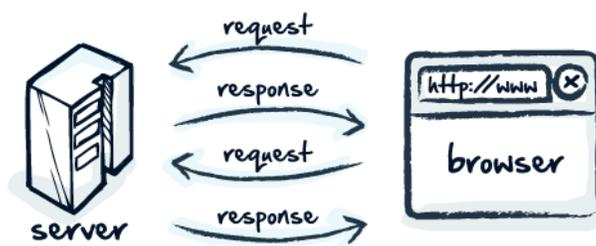


Figure 1: *The Web's Request/Response Model*

For every browser request and subsequent web server response, bandwidth is being used and server resources are being consumed. Even if you are receiving or transmitting very small pieces of data, your servers are still being hit by requests, which they have to process.

The data being transmitted is also sub optimal. Every request being made to the server, as well as every response sent to the user's browser is wrapped inside a group of data that has no real purpose to the user.

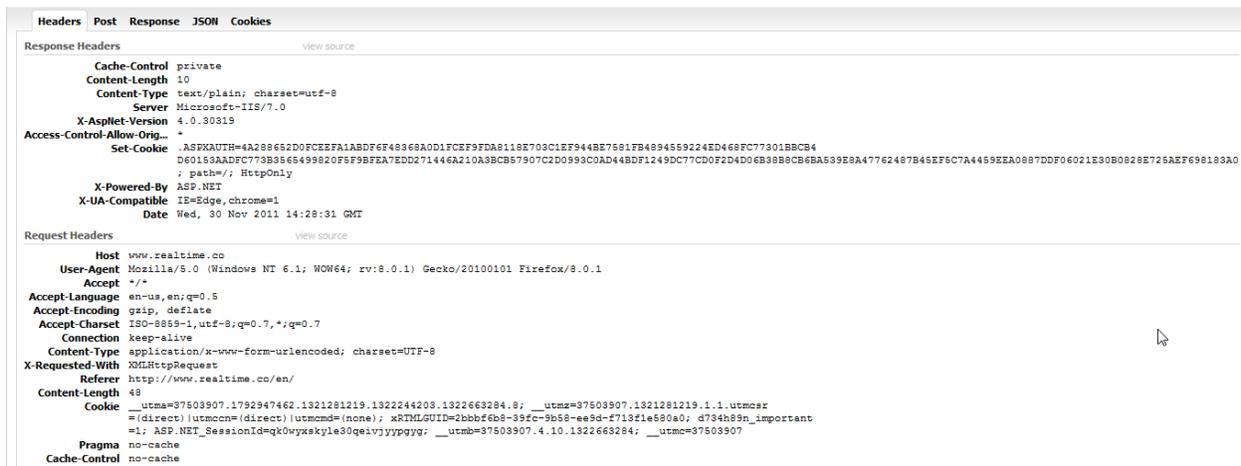


Figure 2: Request and Response Headers

The Near-Real Time Web

Due to the limitations of the Request/Response model, the “real time” we currently experience on websites is nothing more than a simulation of real time. Or “near-real time”.

Users are amazed by web pages that seem to self-update without the need for them to refresh the whole page, having the impression that things are happening in real time, as things happen. In fact, a lot of work is being done in the background, by the user’s browser, the web servers, and anything else in between that is used to carry the information back and forth (modems, routers, connections, servers, etc.). This is the way AJAX works, for example.

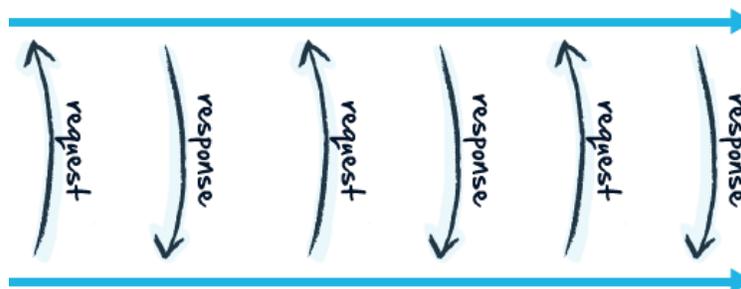


Figure 3: Near - Real Time Traffic Going Back and Forth

Even if there is no new content on the web server, the browser keeps asking for information, consuming bandwidth and using server resources. Also, since requests are made at a set interval, the user will only receive new data – if available at all – at that set interval, meaning that if there’s new information between requests, the user will only receive it on the next request/response iteration. This is hardly real time.

The Realtime® Web

Using Realtime® and the Realtime® Framework, developers can now build truly real time web applications, where the browser no longer takes control of the communication and content can actually be pushed to the visitor – without the excessive need of HTTP requests. Once the web page loads on the browser, Realtime® starts its work, freeing the web server.

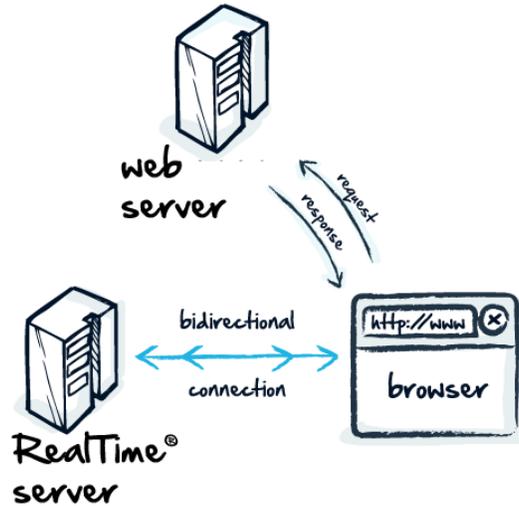


Figure 4: How Realtime® Works

The browser connects to the Realtime® server and, from then on, all the real time messages pass through this new connection and data can be sent from the user to the server and vice-versa. Nevertheless, as opposed to the request/response model, the browser does not need to query the server for data. In fact, this connection can be done in one-way, where data only flows from the server to the user (server push), or only from the user to the server. If necessary, though, it is possible to send and receive data simultaneously, since real time is capable of full duplex.

Since Realtime® data does not go through regular web servers and does not use the HTTP protocol, data exchanged between browsers and the real time servers is also very different. Instead of using the HTTP headers described above, which adds a lot of useless information to the data being exchanged between the server and the browser, Realtime® messages are only appended by 2 bytes, which indicate both the beginning and the end of the data being broadcasted.

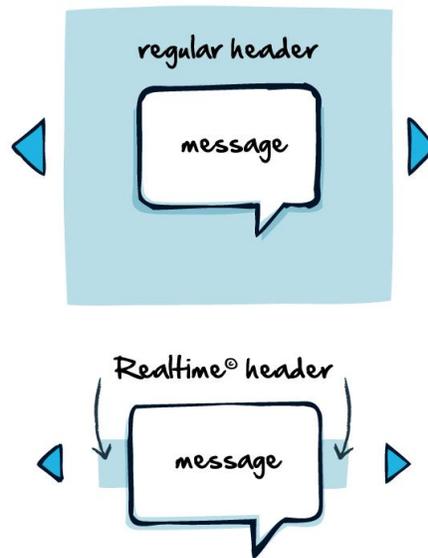


Figure 5: *The Size Difference Between Regular HTTP Messages and Realtime® Messages*

A Realtime® Case Scenario

In order to further understand the advantages of using Realtime® let's pick up an hypothetical case scenario, that probably a lot of companies identify with, in terms of real time information.

The XY Sports website (a fictitious website) offers their visitors a football (soccer for our US friends) live report, in which visitors can read what's happening. XY Sports is an established website and has a lot of users constantly visiting their website. On football match days, that number rises and they get a huge peak of requests on their servers. XY Sports opted to use AJAX to provide this "near-real time" feature. In order to provide a good service, but trying not to bring their systems down, the developers decided that users would make the AJAX request every 30 seconds. To save bandwidth, an empty request is made to the server, through an URL. If there is nothing new, an empty response is sent back to the browser. If there is something new, a very small JSON message is then sent to the user's browser. As we've seen before, every request or response comes with an overhead (the request/response header). Even if the content of the response/request is empty, headers are sent, which means that an average of 2Kb is being transmitted per transmission (4Kb per request/response pair).

For the sake of simplicity, let's assume that XY Sports has a constant amount of users at the website, checking this live report section: 10.000 users. That's 10.000 requests and responses every 30 seconds, or 20.000 per minute. At the end of a 90-minute football match, 1.800.000 requests are made, which means the servers were hit that amount of times and over 7.200.000Kb of data were transferred (a bit over 7.000MB) – just for the request/response headers.

To keep things simple, we'll also assume that there was an event for every minute that has passed, for a total of 90 events. The average message sent back to the users, by the servers would be something like this:

```
{"ttl": "A few bytes here for the title", "desc": "This is the event description, that takes a few more bytes", "tm": "15m"}
```

This adds up some more 124bytes per event, for a total of about 107MB (90 x 124bytes x 10.000 users), for the entire match (excluding the HTTP header – in reality, every event message would be 2.172bytes in size, for a total of 1.8GB).

To resume: without RealTime®, nearly 7GB of data were transferred to serve 107MB of useful information and we had to have our servers processing nearly 2 million requests during 90 minutes of the match. Each event message was 124bytes in average size + 2Kb (2.172bytes total) for the HTTP header.

If XY Sports used Realtime®, there would be no requests from the users. Instead, only the events that occurred during the game would be pushed out to visitors: 124bytes + 2bytes of overhead per message; a total of 108MB.

To summarize:

	AJAX	Realtime®
# Messages/Requests	1.800.000	900.000 ¹
Average message size	2.172bytes	126bytes
Bandwidth used	7.000MB (7GB)	108MB

The difference is obvious: Realtime® delivers a better solution, using much less resources, thus cutting costs. Not only that, it also allows XY Sports to offer a better service, by pushing information to the users as soon as the editorial team releases it. There is no need for the browser to wait until the next server query (up to 30 seconds) to display information to the visitors.

¹ From the Realtime® server to the browser only. No requests are made to the webservers.

The Realtime® Framework

Open Realtime Connectivity (ORTC)

More than just a real time server, ORTC provides a range of production proved advantages that empowers developers with a highly reliable and secure communication layer. For those that already have an IT structure, ORTC also allows developers to work with their real time server or exchange servers quickly and without the need to redo all the communication part of the code.

ORTC works on browsers through the addition of a JavaScript file. Once the file is added to a webpage, a plethora of methods are available to connect/disconnect to a server, subscribe/unsubscribe channels and to send and/or receive messages through those channels.

Layer of Abstraction

ORTC provides a layer of abstraction that protects applications from the ever-changing array of real time server providers. Whenever the chosen provider changes its version or whenever the need to change providers becomes an issue, ORTC shields developers from the need to make massive changes. This allows for faster development or the reduction of development time. Since less code (or practically none) is required, a developer becomes less prone to errors and bugs, which, in turn, also means less time – and money – wasted.

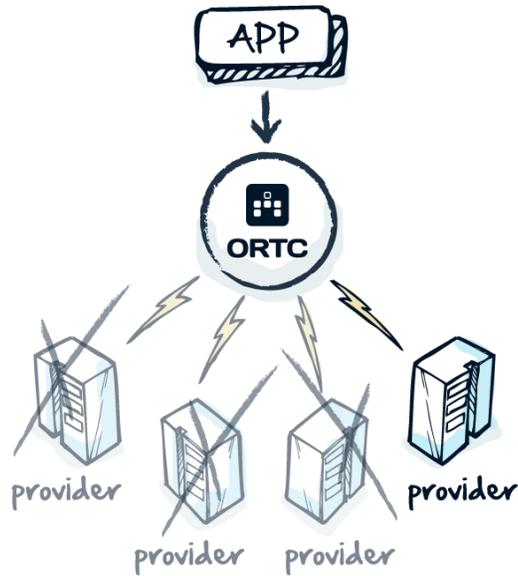


Figure 6: Layer of Abstraction. ORTC Allows for Rapid Provider Change with Minimum Developer Intervention

Multiplexing

Despite one of the main reasons for using Realtime® is the ability to broadcast information to a myriad of users in a fast, low-latency manner, ORTC also offers the ability to communicate with a single-user or a sub-group of users. In a department-type e-commerce website, for example, it's possible to send information (like an ad) to everyone on the website or to target users at a specific department or even to a single user. Of course, the opposite is also possible (get information from the users).

ORTC allows multiplexing, making it possible to transmit information from a single-point to multi-points, the opposite or from multi-point to multi-point. This provides greater flexibility and provides developers the ability to optimize communication between users and servers.

Browsers or any other software (like servers or information routers) can listen to channels and act according, meaning that it's easy to attach foreign processes or systems to a Realtime® powered application, with very little effort from the website or application development team.

Security

We take security very seriously and it was one of the main concerns of ORTC since day 1. A good communication system is only as good as its security and content injection is a real issue with internet-based systems. Users need to feel the information they are sending or receiving is secure. Companies need to make sure no one is tapping into their information and violating their visitors' privacy.

Data goes through our security layer, which allows or denies access to information channels. Even if your provider doesn't offer a security layer, you can always be sure ORTC will be there to protect you, your data and your users.

Security is provided on a channel-base. This means that a user may have read-only access to a channel (will only receive data), may read and write to a channel (may both send and receive data) or no access at all (no data from this channel will be received by the user and he/she won't be able to use the channel to broadcast data).

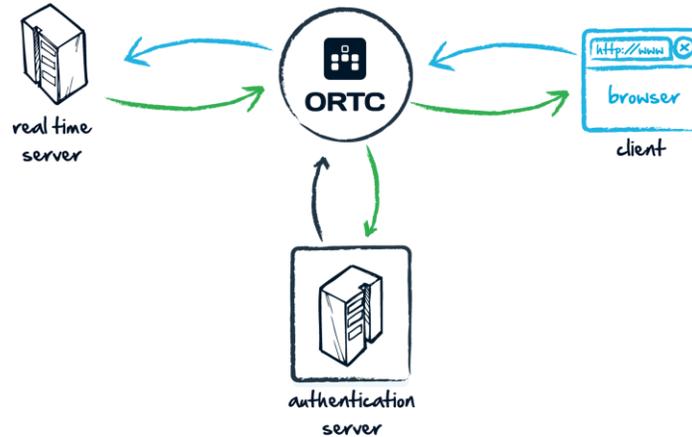


Figure 7: ORTC Security Model

Scalability

Along with security, scalability quickly became a concern, especially because Realtime® provides the means to broadcast information to thousands of users using much less bandwidth and web server resources. Nonetheless, the Realtime® servers must be able to respond reliably and promptly to requests made. This is achieved through the high-scalability architecture or ORTC.

ORTC scales both vertically and horizontally. By scaling vertically, ORTC is able to use the resources of a single Realtime® server to open up internal processes and split the workload among them, instead of using a single thread doing all the work, thus wasting processing power. The limits of a server depend solely on the hardware and not on the software. The better the server, the more amount of connections it will manage to handle.

Of course, every server – even the most powerful one – has its limitations. When servers reach their limit, it is necessary to add more hardware to the equation. ORTC allows you to do just that by allowing horizontal scalability. By using its Load Balancer, ORTC distributes load between two or more servers, vastly increasing the processing power and, therefore, the throughput of the whole Realtime® system. With this, Realtime® allows up to an unlimited number of concurrent users.

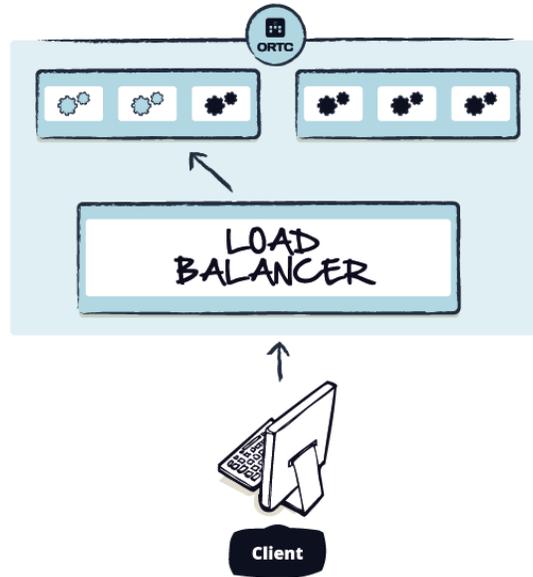


Figure 8: ORTC Scalability: Vertical and Horizontal

Extensible Realtime® Markup Language (xRTML)

xRTML is an HTML-like markup language that allows developers to add real time on their websites. The main purpose of xRTML is to speed up development and offer a level of abstraction from real time communications and DOM manipulation. It comes with a vast array of tags (or controllers) out-of-the-box, which makes it a very powerful tool for developing high interactive websites using Realtime®.

In its main form, xRTML works on browsers, like ORTC, though the inclusion of a JavaScript file. Nonetheless there are also APIs for the most popular server-side languages, such as ASP.Net, PHP, Java, with more planned and under development, which makes it a very convenient cross-platform and powerful language. xRTML works based on tags, just like HTML and everything, from configuration to behaviors are controlled by them.

xRTML controls the flow of information (messages) passing through the Realtime® servers and process that information when necessary, manipulating it and making it interact with the web pages.

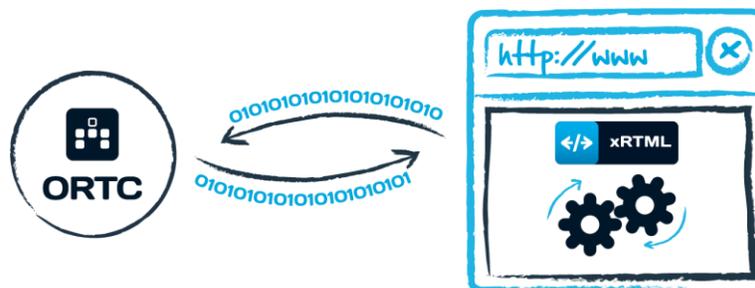


Figure 9: xRTML works inside the browser

An Example

Using xRTML is very straightforward and easy. With only a few minutes' worth of work you can have it running in your website.

Step 1: Add the xRTML Library

Adding the xRTML library is just like adding a regular JavaScript file.

```
<script type="text/javascript" src="http://code.xrtml.org/xrtml.js"></script>
```

Once the library is added, xRTML is ready and will start working once it finds xRTML tags inside the web page.

Step 2: Add a Connection

xRTML already embeds ORTC and all it's needed to do is to configure a connection to the Realtime[®] server and indicate what channel(s) to use.

```
<xrtml:connections>
  <xrtml:connection authenticate="false" url="http://example.ortcserver.com">
    <xrtml:channels>
      <xrtml:channel name="myChannel" />
    </xrtml:channels>
  </xrtml:connection>
</xrtml:connections>
```

Step 3: Add Tags

All is set up and it's all a matter of adding other xRTML tags to the page. The following example is for a Repeater Tag. This particular tag picks up the template code (in bold) and replaces the placeholders (between brackets) with the content that comes from Realtime[®] messages. It then adds the result to a given object inside the webpage.

```
<xrtml:repeater target="#list1" index="begin" removeindex="end" maxitens="5">
  <xrtml:template><!--<li> [xRTML:name] </li>--></xrtml:template>
  <xrtml:triggers>
    <xrtml:trigger name="myTrigger"></xrtml:trigger>
  </xrtml:triggers>
</xrtml:repeater>
```

The list in cause would be something like the one below:

```
<ul id="list1">
  <li>List item 1</li>
  <li>List item 2</li>
  <li>List item 3</li>
</ul>
```

Cross-Browser and Cross Platform

xRTML works on most browsers and platforms (desktop computers and mobile devices) without the need of plugins and any special care from the developers. In fact, it works even on older browsers such as Internet Explorer 5.5. Of course that newer browsers provide better support and the latest versions allow to unfold the full power of xRTML and the Realtime[®] Framework.

Further Reading

Realtime.co

www.realtime.co is the website for all Realtime® products. Here you will find commercial information on either the Realtime® Framework or other Realtime® products such as Power Marketing or the RT AD Server.

xRTML.org

At www.xrtml.org you can find technical information about xRTML, the Realtime® markup language that provides developers with the tool to create their Realtime® applications or adapt Realtime® to the ones they already have, with very little effort.

What Realtime Means to the Future of the Web

Mr. Andy Batkin (CEO at Social TV Summit), talks about Realtime® and its importance.

- http://www.realtime.co/en/testimonials/andy_batkin_225.html