Title: SYSTEM AND METHOD FOR TELEVISION ENHANCEMENT

Set Top Box Models
- Mitsubishi WB-2001
- Philips Magnavox MAT972
- Philips Magnavox MAT976
- RCA RW2110
- Sony INT-W250
- Sony INT-W200

HTML Support
- HTML 1.0
- HTML 2.0
- HTML 3.2
- Frames Compatibility
- JavaScript 1.2

Image Support
- GIF89a animation
- JPEG
- Progressive JPEG
- PNG
- TIFF/G3 Fax in Email
- X bitmap
- Macromedia™ Flash 1.0
- Macromedia™ Flash 2.0
- Macromedia™ Flash 3.0
- Transaction

Abstract: A text based script file describing enhancements is parsed to produce platform dependent enhancement files that may be employed to produce enhancements on a set topbox, enhanced television, or computer display. A script file may be of XML format and apsers may be an XSL translator. A parser may import HTML and JavaScript from other applications. A parser may support a media player for emulation to view video and enhancements. Trigger data for rendering enhancements may be formatted into a javascript array. Trigger events may be employed to replace graphic pointers or text values. Multiple parsers, each supporting a specific platform, allow a single script file to be employed across multiple platforms. New platforms or versions of platforms are supported through new or updated parsers.
CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— without international search report and to be republished upon receipt of that report

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.
SYSTEM AND METHOD FOR TELEVISION ENHANCEMENT

Cross Reference to Related Applications

This application is based upon and claims priority of United States provisional application number 60/227,063 entitled "A DATA DRIVEN SYSTEM AND METHOD FOR DISTRIBUTION OF INTERACTIVE CONTENT TO MULTIPLE TARGETTED PRESENTATION PLATFORMS", filed August 21, 2000 by Steve O. Markel, the entire disclosure of which is herein specifically incorporated by reference for all that it discloses and teaches.

Background

a. Field

The present disclosure relates to interactive and enhanced television and, more particularly, to a method and system that produces enhanced content that may be employed across a plurality of platforms without re-editing. In greater detail this disclosure discusses systems and methods for distribution of interactive content to multiple targeted presentation platforms.

b. Description of the Background

A television program may be accompanied by additional information employed to enhance the program or to provide viewer interaction. Enhancements have historically included closed captioning and multilingual support. Advances in networking, computer systems, and video production have increased the number and types of enhancements that may be provided with a program or advertisement. Enhancements may include stock updates, news stories, Internet links, weather forecasts, bulletins, statistics, trivia, and other information. For example, a football game may include icons allowing viewing of team players, statistics, trivia and other information such as upcoming games. Further, the advent of set-top-boxes, as may be used in cable and satellite television systems, allows enhancement information to be presented in new ways, such as screen overlays and in windows, for example.
Enhanced television content may employ a combination of HTML (hypertext markup language), JavaScript, Java and other formats common to Internet page display. An enhanced display may comprise text, icons, graphics and images placed at locations on or in proximity to the television image. To produce an enhanced display, an author must create a file identifying each displayed element (such as text, icons, graphics and images), the location where each element is displayed and the time at which the element may be displayed. Due to numerous differences between presentation platforms, such as set top boxes, satellite receivers, computers, or interactive televisions, for example, content providers have historically been required to select a specific platform in the development of an enhancement application. In order to provide support for each additional platform, the interactive content provider must introduce potentially significant modifications to the existing application, resulting in the ongoing maintenance of multiple code bases, and adding to the time and cost required producing enhanced page layouts for multiple platforms.

Additionally, previous methods employed to enter parameters required to generate and position the elements comprising the layout of enhanced pages have involved significant manually entry. Manual editing of an enhancement file may also introduced unintended changes such that enhancements are not uniform across platforms. Therefore a new method of creating enhanced content that allows utilization across multiple platforms and provides an accurate preview of enhancements is needed.

**Summary of the Invention**

The present invention overcomes the disadvantages and limitations of the prior art by providing a system and method that parses a text based script enhancement file to provide emulation of enhancements and to provide output of platform specific enhancement files. The enhancement file, which may employ an XML format, contains a description of enhancements including element position, attributes, triggering and linkage. Linkage associates a file with an element such that a user may select a linked object to access websites, launch other applications, or to perform other tasks. Parsing the enhancement file produces an output file targeted to a specific platform. The platform may comprise a
set top box, interactive television or computer display. A file parsed for computer display
may be employed for emulation and preview of enhancements and may contain functions
for control of a browser media player.

The invention therefore may comprise a method for creating a television
presentation enhancement comprising accessing a platform independent enhancement file
containing elements and attributes of the elements, applying a first parsing script to the
enhancement file to produce a first output file that may be viewed using a web browser
and media player, and applying a second parsing script to the enhancement file to
produce a second output file that may be viewed with a set top box.

An enhancement file of XML format may be parsed using XSL (Extensible
Stylesheet Language) scripts. In the present invention, an XML file with tags for
administrative information, layout information, and trigger information is employed. The
XML file may be parsed to produce an output file containing HTML and JavaScript code
wherein the version of HTML and JavaScript reflect the level of support provided by the
target platform. Further, the output file may be formatted for a mode of enhancement
transport. In a first mode of transport, enhancements and triggers are supplied in
conjunction with a video program. In a second transport mode, triggers and a locator,
such as a URL, are provided in conjunction with a video program and the platform
employs the locator to access enhancement information.

The invention may further comprise a system for developing television
enhancements comprising a computer; a database; a web browser; and a parser operable
to parse a platform independent enhancement file contained in the database and to
produce an output that may be viewed employing the browser.

Advantageously, the invention provides viewing and emulation of enhancement
files employing a personal computer or similar equipment. This allows a team of
developers and reviewers to be physically separated, and allows enhancement customers
(such as advertisers) to preview material by simply accessing a website.
Description of the Figures

In the figures,
- Figure 1 depicts html and image support for a group of commercially available set top box products.
- Figure 2 depicts the software environment of the present invention.
- Figure 3 is an overview flowchart of parsing an XML file to emulate and preview enhancements.
- Figure 4 depicts a first flowchart of part of a parsing process.
- Figure 5 depicts a second flowchart of a parsing process.
- Figure 6 depicts a third flowchart of a parsing process.
- Figure 7 depicts a computer display of an enhancement.

Detailed Description of the Invention

Enhanced television content is typically presented using a combination of HTML, JavaScript, Java and other web technologies. The level of support for these technologies varies by the targeted presentation platform, including the combination of client hardware, operating system, web browser and add-on software. A presentation platform comprises a set top box, interactive television, computer, or other system operable to receive television signals and to process HTML and other code and to produce a display comprising a television image and enhancements. Capabilities vary depending the specific platform. Certain functions may or may not exist, or may be optimized on a given platform through the use of custom features. Variants include screen size and resolution, acceptable color combinations, graphics support, and version of HTML or JavaScript, for example. Providing concurrent support for the Internet or wireless handheld devices introduces additional requirements and dependencies. The present invention overcomes the difficulties of supporting multiple platforms, each having a specific set of capabilities, by employing a platform independent text based script file that completely defines the enhancement assets, their location and other attributes, as well as the triggering information. The text based script file is then translated by parsing software to produce platform dependent files comprising HTML and JavaScript code tailored to
the specific platform. The present invention also allows translation targeting a web browser and a media player, providing emulation and preview of authored enhancements.

Figure 1 depicts HTML and image support for a group of commercially available set top box products. Set top box models 100 provide HTML support 102 and image support 104. HTML support 102 lists support for html 1.0, 2.0, and 3.2 versions. A limitation of HTML is that some versions lack downward compatibility. For example, HTML versions 4 and higher do not support all the tags of HTML 3.2. Figure 1 serves to illustrate that an HTML based author for creating enhanced content would not be able to support a wide range of target platforms. The present invention overcomes the disadvantages of HTML based authoring by employing an authoring tool that generates an XML file that may be parsed using XSL scripts for each platform type to produce HTML code and JavaScript suitable for each platform.

Figure 2 depicts the environment of the present invention. Environment 200 comprises database 208 containing user and project administration information 202, page layout information 204 and trigger creation information 206. XML file 210 is created using information from database 208. Import XSL's 212 may be employed to translate HTML and JavaScript into XML file 210. Emulator XSL 214 provides translation of XML file 210 into HTML and JavaScript, plus provides media player controls for emulation and preview. STB Agnostic Sniffer XSL 216 includes platform query routines to determine the type of platform requesting enhancement information. WebTV™ XSL 218 provides translation compatible with WebTV platforms. AOLTV™ XSL 220 provides translation compatible with AOL TV platforms. Triggers XSL 222 provides triggers that may be transmitted with a television presentation that may be used to synchronize display of enhancements. Enhancement information may be transmitted with the television presentation, or may be accessed by the platform in response to the trigger information. Translated files may be stored in server 224.

Figure 3 is an overview flowchart of parsing an XML file to emulate and preview enhancements. Process steps shown in figure 3 are described in greater detail in following figures. Parsing process 300 starts with step 302 where comments are inserted that indicate the project name, page names, date, time, and other information. This information will form in part, the header of the resultant HTML/JavaScript file produced.
by the parsing process. At step 304, a list of elements comprising an enhancement is
scanned and checked for a JavaScript tag. If a JavaScript tag is found, the JavaScript is
placed into the output file. At step 306, trigger tags are extracted, trigger data is sorted
and a JavaScript trigger array is created that comprises time, element, and new element
value. This array is referenced during emulation. At step 308, the list of elements is
scanned and a function to change the text value is generated for elements with a text area
tag. This function allows text in a text area to be changed, such as response to a trigger
event. At step 310, the list of elements is scanned and a function to change the source
value is applied to graphics elements. At step 312, code that references the trigger array
and individual function is inserted. At step 314, code is inserted that is executed when the
browser window is opened. At step 316, the list of elements is scanned and if an imported
HTML tag element is found, the value is extracted and placed in the output file. At step
318, code supporting a media player including stop, go, pause, and timer is written to the
output file. At step 320, a media player object is placed in the output file if a 'TV' object
is present in the XML source file. Graphics elements are placed in the output file with
'img' tags and text elements are placed in the output file with text tags. A web browser
may access the output file created by the above parsing process and the enhancement may
be displayed. If the enhancements are related to a television image, the media player
module allows viewing in conjunction with display of a video image. The module allows
the media player to present a video sequence, along with enhancements. The media
player may be paused, stopped, started, or the user may go to a specific frame or display
time. The following figures provide a more detailed description of the steps employed in
parsing an XML source file.

Figure 4 depicts a first flowchart of part of a parsing process. Process 400 starts at
step 402 where a looped process for each page of the XML file begins. Page loop 404
provides a return path for the process when an additional page or pages remain. At step
406, <HTML> and <HEAD> open tags are written. At step 408 a <TITLE> tag is written
with the page name. At step 410, ownership and contact comment information may be
written. At step 412, a project name comment may be written. At step 414, a page name
comment may be written. At step 416, an author comment may be written. At step 418, a
comment indicating the date the XML file was authored may be written. At step 420, a
notes comment may be written. At step 422 a JavaScript tagging process begins. At step 424, an element is accessed from the XML file and is checked to determine if the element is of import type. If the element is not of import type, processing returns to step 422 where another element is accessed. If all elements have been accessed, processing continues at step 434. If the result of step 424 is that the accessed element is of import type, step 426 writes a language specification indicating that the script language is JavaScript. At step 428, the contents within the 'js' tag of the XML file are written. At step 430, a "</script>" closing tag is written. At step 432, processing continues to step 422 to access additional elements. If all elements have been accessed, processing continues at step 434. At step 434 JavaScript variable statements for trigger emulation are written. At step 436, an opening tag for an array of triggers is written. At step 438 a processing loop is started. Step 438 accesses trigger information, ordered by time, each time the loop is executed. At step 440, the trigger time, multiplied by 1000, is written. At step 442, the element name affected by the trigger is written. At step 444, parameters associated with each trigger are written. Step 446 checks if additional trigger information may be accessed for the current page. If additional information may be accessed, processing continues at step 438. When all trigger information has been accessed, processing continues at step 448 where a value indicating the end of the trigger array is written. Step 450 leads to the steps shown in figure 5.

Figure 5 depicts a second flowchart of a parsing process. Step 502 is a continuation from the steps shown in figure 4. At step 504, a processing loop accesses each element of the current page. Step 506 checks if the element is a text element. If the element is a text element, step 508 writes a function for changing the text value within the text area. Processing then continues with step 514. If step 506 determines that the element is not a text element, step 510 checks if the element is a graphics element. If the element is a graphics element, step 528 writes a function for changing the source value within an image field. Processing then continues with step 503. If step 510 determines that the element is not a graphic element, processing continues at step 514. At step 514, processing continues at step 504 if additional elements remain. If all elements have been accessed, processing continues at step 516. Step 516 writes a script closing tag. Step 518 then writes a script language tag indicating that the script language is JavaScript. Step
522 writes setup variables for a media player. Step 524 then converts numeric seconds to an ASCII string. Step 526 writes a function to handle start, stop and pause controls for the media player. Step 528 writes a function for time display. Time display may be used to indicate the time of a video sequence being shown by the media player. At step 530, event code for selection of the media player 'go' button is written. At step 532, event code the selection of the media player 'stop' button is written. At step 534, event code for selection of the media player 'preview' button is written. At step 536, event code is written that is executed when the emulation window is opened. At step 538 a closing script tag is written. At step 540, a closing head tag is written. Step 542 writes a 'body' tag with an event handler for window loading. Step 544 leads to the steps shown in figure 6.

Figure 6 depicts a third flowchart of a parsing process. Step 602 is a continuation from the steps shown in figure 5. At step 604, a processing loop accesses each element within each page. Step 606 checks if the element is of import type. If the element is not of import type, processing continues at step 604 where the next element is accessed. If all elements have been accessed, processing continues at step 616. If step 606 determines that the element is of import type, step 608 writes a division tag (<div>) and writes body code comprising element name, absolute position, top left position and z index. The z index value may be employed to control the order in which elements are rendered, causing one element to appear on top of another element. Step 610 writes the contents of the 'htmlBody' tag. Step 612 then writes an end <div> tag. At step 614, if all elements have not been accessed, processing continues at step 604. If all elements have been accessed, processing continues at step 616. Step 616 writes a division tag for media player positioning buttons. Step 618 writes <table> code with positioning buttons. Step 620 writes an end <div> tag, demarking the division started at step 616. At step 622, a processing loop accesses each element within each page. Step 624 checks if the element type is graphic. If the element is a graphic element, step 626 writes a division tag for html body code comprising element name, absolute position, top left position and z index. Step 628 checks if a URL (Universal Resource Locator) exists for the element. If a URL exists, step 630 writes a URL html tag. Step 632 then writes an 'img' tag with element name, border=0, and source. Source is the address of where the graphic element is stored. Processing then continues at step 644. If step 628 determines that a URL does not exist
for the element, step 632 writes an ‘img’ tag with element name, border=0, and source. 
Processing then continues at step 644. If step 624 determines that the element is not a 
graphics element, processing continues at step 634. Step 634 checks if the element is of 
type ‘tv’. If the element is of type ‘tv’, step 636 writes a division tag for HTML body code 
comprising element name, absolute positioning, top left position, and z index. Step 638 
writes code to embed a media player. Processing then continues at step 644. If step 634 
determines that the element is not a ‘tv’ type element, processing continues to step 640 
where a check is performed if the element is a text area element. If the element is a text 
area element, step 642 writes a division tag for HTML body code comprising element name, 
absolute positioning, top left position, z index, font, color, and point size. Processing then 
continues at step 644. If step 640 determines that the element is not a text area element, 
processing continues at step 644. At step 644 processing loops back to step 622 if 
all elements remain that have not been accessed. Otherwise, processing continues at step 
646. If pages remain that have not been accessed, processing continues to step 648 
where the process loops back to step 404 of figure 4 to access the next page. If all pages have 
been accessed, step 650 writes closing <body><html> tags and the process ends at step 
652.

Figure 7 depicts a computer display of an enhancement. A text based script file, 
employing an XML format, describing the elements employed to create the enhancement 
depicted in figure 7 is listed in Appendix A. Appendix B lists an HTML file with 
JavaScript that has been produced from the code of Appendix A through the process 
described in figures 3 to 6. Appendix B includes section identifiers that relate the code 
sections to steps shown in figure 3.

In operation, a user logs into an editing system, creates a project, and then lays out 
enhanced content pages and creates triggers for those pages. A database stores project 
information that comprises pages and triggers and may include project name, author date 
and other information. Information in the database is employed to create a text based 
script file that describes each element, its attributes, its layout and triggering of the 
element. In one embodiment of the present invention, an XML file is employed. This file 
is output platform independent and completely defines the assets, their location and other 
attributes, as well as the triggering information necessary for the enhanced content
project. The XML file is then processed using one or more XSL rule based parsers that "translate" the XML file into another format, such as HTML 4.0 and Javascript 1.2, for example. As depicted in figure 2, XSL parsers are employed for emulation and for cresting platform specific output files. Parsing of the enhancement file for a particular platform may include translation of color values. Some platforms do not display pure colors and as such a lookup table or translation algorithm may be employed to check a color value and to alter the color value. The process shown in figures 3 to 6 produces an HTML and Javascript output that may be run on an industry standard web browser and media player such as Microsoft Internet Explorer and Windows Media Player, both from Microsoft Corporation. Additional information regarding XSL may be obtained from the following books:

Title: Professional XSL
Authors: Kurt Cagle et al.
Publisher: Wrox Press Inc;
ISBN: 1861003579

Title: XSL Companion, The
Author: Neil Bradley
Publisher: Addison-Wesley Pub Co;
ISBN: 0201674874

The foregoing description provides a system and method that translates a platform independent enhancement file into platform dependent files without needing to change the authored enhancements, saving time and money and providing a uniformity of enhancement across multiple platforms. The steps shown in the figures need not be performed in the exact order shown. An XML file format has been employed in the described embodiment. Other formats, both public and proprietary, may be employed to describe enhancements and attributes of the elements comprising enhancements.

Enhancement output files allow enhancement of a television broadcast that may employ various methods of delivering enhancement data. A first method transfers enhancement
data as part of the broadcast. A second method transfers a trigger and locator with the broadcast, and the set top box, or other platform, employs the locator to access enhancement information. The second method of transfer may further comprise receiving information identifying the type of platform requesting enhancement data (when the platform accesses the locator), and providing enhancement information suited to the requesting platform. A new or modified parser may be employed to support new platforms, or new versions of platforms. The new or modified parser then may be applied to a plurality of source enhancement files that need not be modified. In this manner, the present invention provides cost savings in supporting new platforms or new versions of platforms.

The foregoing description of the invention has been presented for purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise form disclosed, and other modifications and variations may be possible in light of the above teachings. The embodiment was chosen and described in order to best explain the principles of the invention and its practical application to thereby enable others skilled in the art to best utilize the invention in various embodiments and various modifications as are suited to the particular use contemplated. It is intended that the appended claims be construed to include other alternative embodiments of the invention except insofar as limited by the prior art.
Appendix A

```xml
<?xml version="1.0" ?>
<!-- Viziware XML Schema 1.1 -->
<!-- <project> 
<emulate>True</emulate>
<useWMP>False</useWMP>
<pjName>demo project</pjName>
<date>8/11/2000 1:14:09 PM</date>
<author>Steve Markel</author>
<canvas>pal</canvas>
<notes>simple one page enhancement</notes>
</project>

<pages>
<page>
<pgName>index</pgName>
<element>
<elName>tv</elName>
<type>tv</type>
<src />
<top>0</top>
<left>0</left>
<height>392</height>
<width>523</width>
<zOrder>-1</zOrder>
</element>
</page>
</pages>
```

```xml
<element>
<elName>nav</elName>
<type>graphic</type>
<src>C:/viziware/betaGraphics/nav.jog</src>
<top>0</top>
<left>519</left>
<height>480</height>
<width>121</width>
<zOrder>1</zOrder>
</element>
```

```xml
<url />
</font />
</color />
```
<size />
<rows>-1</rows>
<cols>-1</cols>
</element>
- <element>
  <elName>button1</elName>
  <type>graphic</type>
  <src>C:/viziwork/betaGraphics/button1.jpg</src>
  <top>108</top>
  <left>530</left>
  <height>20</height>
  <width>97</width>
  <zOrder>3</zOrder>
  <url />
  <font />
  <color />
  <size />
  <rows>-1</rows>
  <cols>-1</cols>
</element>
- <element>
  <elName>button2</elName>
  <type>graphic</type>
  <src>C:/viziwork/betaGraphics/button2.jpg</src>
  <top>144</top>
  <left>526</left>
  <height>25</height>
  <width>97</width>
  <zOrder>4</zOrder>
  <url />
  <font />
  <color />
  <size />
  <rows>-1</rows>
  <cols>-1</cols>
</element>
- <element>
  <elName>button3</elName>
  <type>graphic</type>
  <src>C:/viziwork/betaGraphics/button3.jpg</src>
  <top>187</top>
  <left>525</left>
  <height>19</height>
  <width>97</width>
<zOrder>5</zOrder>
<url />
<font />
<size />
<rows>-1</rows>
<cols>-1</cols>
</element>

- <element>
  <elName>button4</elName>
  <type>graphic</type>
  <src>C:\viziworx\betaGraphics\button4.jpg</src>
  <top>221</top>
  <left>528</left>
  <height>42</height>
  <width>97</width>
  <zOrder>6</zOrder>
  <url />
</element>

- <element>
  <elName>text</elName>
  <type>graphic</type>
  <src>C:\viziworx\betaGraphics\text.jpg</src>
  <top>391</top>
  <left>0</left>
  <height>89</height>
  <width>521</width>
  <zOrder>2</zOrder>
</element>

- <element>
  <elName>textarea8</elName>
  <type>ta</type>
  <src />
<triggers>
  <trigger>
    <time>2</time>
    <prop>txt</prop>
    <elName>textarea9</elName>
    <param>the first prompt</param>
  </trigger>
  <trigger>
    <time>4</time>
    <prop>txt</prop>
    <elName>textarea9</elName>
    <param>the second prompt</param>
  </trigger>
  <trigger>
    <time>6</time>
    <prop>src</prop>
    <elName>ad</elName>
    <param>C:/ViziworxTestfiles/betaGraphics/amazon.jpg</param>
  </trigger>
  <trigger>
    <time>8</time>
    <prop>txt</prop>
    <elName>textarea9</elName>
    <param>the final prompt</param>
  </trigger>
</triggers>
</page>
</pages>
Appendix B

This appendix shows code generated from the XML file listed in Appendix A employing a parser of the present invention. Steps shown in bold reference steps shown in figure 3.

[Step 302]

```html
<html>
<head>
<title>index</title>
</head>
<body>
 ... 
</body>
</html>
```

[Step 304]

```javascript
var timerDelta=50;
var startCount=0;
var timerID;
var ms = 0;
var lnx = 0;
var paused = 0;

var aTriggers = new Array()
2*1000, "textarea8('the first prompt')",
4*1000, "textarea8('second prompt')",
6*1000, "ad('C:\viziwork/betaGraphics/amazon.jpg')",
8*1000, "textarea8('final prompt')",
99999999, "");

[Steps 308 and 310]

function fnav(theImg) { document.all["nav1"].src = theImg; }

function fbutton1(theImg) { document.all["button1"].src = theImg; }

function fbutton2(theImg) { document.all["button2"].src = theImg; }

function fbutton3(theImg) { document.all["button3"].src = theImg; }

function fbutton4(theImg) { document.all["buttons"].src = theImg; }
```
function ftext(theImg) { document.all["text"][src = theImg; }

function ftextarea8(theTxt) { window textarea8.innerText = theTxt; }

function flocallogo(theImg) { document.all["localogo"]); src = theImg; }

function fadeImg() { document.all["ad"][src = theImg; }

</script>

[Step 312]

<script language="javascript"> <|--
var ms = 0;
var state = 0;
function __secs2asc(t) {
    var tSecs = Math.floor(t/1000);
    var hrs = Math.floor(tSecs/3600);
    var mins = Math.floor((tSecs-(hrs*3600)) / 60);
    var secs = tSecs-(hrs*3600) + (mins*60));
    var ms = t % 1000;
    if(hrs>23) return "
    if(hrs < 10) hrs = "0" + hrs;
    if(mins < 10) mins = "0" + mins;
    if(secs < 10) secs = "0" + secs;
    if(ms < 10) ms = "00" + ms;
    if(ms < 100) ms = "00" + ms;
    if(ms == 0) ms = "000";
    return hrs + ":" + mins + ":" + secs + "," + ms;
} function __startStop() {
    if (state == 0) {
        ms = 0;
        state = 1;
        now = new Date();
        then = new Time();
        then.setTime(then.getTime() = ms);
        document.WMPPlay.Play();
        window.__frmTr.__cmdWPGo.value="Pause";
    } else {
        state = 0;
        now = new Date();
        ms = now.getTime() - then.getTime();
        window.lblTime.innerText = __secs2asc(ms);
        document.WMPPlay.Pause();
        window.__frmTr.__cmdWPGo.value=" Go 
    }
}
function __timeDisplay() {
    timerID = setTimeout("__timeDisplay();", 50);
    if (state == 1) {
        now = new Date();
        ms = now.getTime() - then.getTime();
        window.lblTime.innerText = __secs2asc(ms);
function __cmdMFPGo_onclick() {
    if(timerID) clearTimeout(timerID);
    __startStop();
    btnGo_onclick();
}

function __cmdMFPStop_onclick() {
    document.WMPlay.Stop();
    document.WMPlay.CurrentPosition = 0;
    state = 0;
    window.lbTime.innerText = __secs2asc(0);
    window.__frmTr.__cmdMFPGo.value=" Go ";
}

function Preview() {
    var func, ps, pe, param;
    if (aTriggers[iaT] <= ms) {
        func = "f" + aTriggers[iaT+1];
        eval(func);
        iaT = iaT + 2;
    }
    ms = ms + timerDelta;
    timerID = setTimeout("Preview()", timerDelta);
}

function btnStop_onclick() {
    clearTimeout(timerID);
    pauseGo = 0;
}

function btnPause_onclick() {
    if (pauseGo == 0) {
        btnStop();
        pauseGo = 1;
    } else {
        Preview();
        pauseGo = 0;
    }
}

function btnGo_onclick() {
    var offset;
    pauseGo = 0;
    for (ms=0, iaT=0; aTriggers[iaT] < ms; iaT=iaT+2) {}
    Preview();
}

function window_onload() {
    var p0 = document.body.innerHTML.indexOf("<!--__stbody-->");
    if (p0 > -1) {
        var p1 = document.body.innerHTML.indexOf("<![CDATA[","p0)+9;
        if(p1 > 8) { //if not, there was no imported html
            var s = "<DIVx</DIVx></DIVx>;";
        var p2 = document.body.innerHTML.indexOf(s);
var cl = p2-p1;
document.body.innerHTML =
document.body.innerHTML.substr(p1, cl) + "</div>" +
document.body.innerHTML.substr(p2+s.length);
}
}

window.__frmTr__cmdMPGo.disabled=true;
window.__frmTr__cmdMPStop.disabled=true;
document.wmPlay.FileName = "demo.asf";
document.wmPlay.ShowControls = false;
document.wmPlay.AutoStart = false;
window.__frmTr__cmdMPGo.disabled=false;
window.__frmTr__cmdMPStop.disabled=false;

-->
</script>
</head>

[Step 316]

-- none -->

[Step 316]

<body LANGUAGE="javascript" onload="return window_onload()">

</body>

[Step 316]

<div id="divTr" style="position:absolute; top:500; left:0; z-index:0">
<form id="__frmTr" name="__frmTr">
<table border="0">
<tr>
<td>
<input type="button" value="Go" id="__cmdMPGo" LANGUAGE="javascript" onclick="return __cmdMPGo_onclick()" />
<input type="button" value="Stop" id="__cmdMPStop" NAME="__cmdMPStop" LANGUAGE="javascript" onclick="return __cmdMPStop_onclick()" />
</td>
</tr>
<tr>
<td style="COLOR: #ff6000; FONT-FAMILY: Verdana; FONT-SIZE: 10pt; FONT-WEIGHT: bold; BACKCOLOR: #000000">
LABEL id="lblTime" STYLE="00:00:00.000"</LABEL>
</td>
</tr>
</table>
</form>
</div>

[Step 320]

<object classid="CLSID:22D6F112-B0F6-11D0-94AB-0000C74C7E95" codebase="http://activex.microsoft.com/activex/controls/wmplayer/en/nsmp2inf.cab#Version=6,4,5,715" type="application/x-oleobject" id="WMPlay" standby="Loading Microsoft Windows Media Player...">
<embed type="application/x-mplayer2" filename="demo.asf" displaysize="4" name="WMPlay" width="523" height="392" />
</object>

</div>

</div>

</div>

<style type="text/css" media="screen">
	div#nav { position: absolute; top: 0; left: 519; z-index: 1; }
	img { border: 0; }
</style>

<script type="text/javascript" src="C:\viziwork\betaGraphics/nav.jpg"></script>
Preview Text Area
Claims

What is claimed is:

1. A method of television enhancement produced by the steps of:
   using a platform independent television enhancement file comprising elements and
   attributes of said elements; and
   parsing said television enhancement file to produce an output file that may be
   viewed with a specific platform.

2. The method of claim 1 wherein said platform independent television is contained in a
   database and in said using step, the term using means accessing, said method further comprising
   the step of saving said output file.

3. The method of claim 1 wherein said method for creating said television enhancement is a
   presentation, said term using means accessing, said parsing step comprising:
   applying a first parsing script to said enhancement file to produce a first output
   file that may be viewed using a web browser and media player; and
   applying a second parsing script to said enhancement file to produce a second
   output file that may be viewed with a set top box.

4. The method of claim 1 wherein in the step of using, the term using means editing such
   that said editing specifies said elements and attributes, said method being for creating a television
   enhancement presentation, said parsing step comprising:
applying a first parsing script to said enhancement file to produce a first output file that may be viewed using a web browser and media player; and
applying a second parsing script to said enhancement file to produce a second output file that may be viewed with a set top box.

5. The method of claim 3 wherein said enhancement file further comprises:
XML compliant tags for elements, triggers, and administrative information comprising enhancement file name and enhancement file creation date.

6. The method of claim 3 wherein said television enhancement file is a text file.

7. The method of claim 3 further comprising:
   displaying said first output file in a browser window.

8. The method of claim 3 wherein said step of applying a second parsing script further comprises:
   specifying an HTML version for said second output file.

9. The method of claim 3 wherein a link is associated with at least one of said elements.

10. The method of claim 3 wherein said attributes of said elements further comprises:
a z order value for at least one of said elements.
11. The method of claim 3 wherein said first parsing script is an XSL transformation file.

12. The method of claim 3 wherein said second parsing script is an XSL transformation file.

13. The method of claim 3 further comprising:
   storing said second output file.

14. The method of claim 3 wherein said second parsing script imports HTML code into said second output file.

15. The method of claim 3 wherein accessing said enhancement file further comprises:
   accessing a database.

16. The method of claim 3 wherein said second parsing script is operable to translate a color value.

17. The method of claim 1 or 2 wherein said enhancement file is XML compliant.

18. The method of claim 17 wherein said step of parsing further comprises:
   applying an XSL transformation to said enhancement file.

19. The method of claim 17 wherein said step of parsing further comprises:
   translating a color value associated with one of said elements.
20. A system for developing television enhancements comprising:

   a computer;
   a database;
   a web browser; and
   a parser operable to parse a platform independent television enhancement file contained in said database and to produce an output file that may be viewed employing said browser.

21. The system of claim 20 further comprising:

   a parser component operable to enable display of a media player if said television enhancement file contains an element representative of a television image.

22. The system of claim 20 wherein said parser is operable to create an output file for a specific platform.

23. The system of claim 20 wherein said parser is operable to translate a color value associated with an element contained in said television enhancement file.

24. A parser for producing a platform specific television enhancement file comprising:

   a function to access a platform independent television enhancement file comprising project information, a description of an element, the position of said element, and a time at which said element may be rendered.
   a function to create an HTML header containing said project information;
Figure 1

Set Top Box Models

Mitsubishi WB-2001
Philips Magnavox MAT972
Philips Magnavox MAT976
RCA RW2110
Sony INT-W250
Sony INT-W200

HTML Support

HTML 1.0
HTML 2.0
HTML 3.2
Frames Compatibility
JavaScript 1.2

Image Support

GIF89a animation
JPEG
Progressive JPEG
PNG
TIFF-G3 Fax in Email
X bitmap
Macromedia™ Flash 1.0
Macromedia™ Flash 2.0
Macromedia™ Flash 3.0
Transaction
Figure 3

1. Print Comments
2. For each imported js element, insert js code
3. Create and insert variables into array
4. For each text area element, create and insert text value function
5. For each graphic element, create and insert graphics function
6. Insert javascript code
7. Insert Window onload event handler
8. For each imported html element, insert html code
9. Place the emulation step and go buttons onto page
10. For each element...
11. Is type="input"?
12. Is type="text"?
13. Is type="text area"?
14. Insert window media player object into html stream
15. Insert <img> tag with params into stream
16. Insert text tags into stream
17. End of elements?

END

JavaScript

HTML
Figure 7