MediaMan Theme Development Guide

This document describes the specification of the theme feature in MediaMan 3. This document contains no specific copyrights and is released to Public Domain. However, the content of the sample files to be discussed contains copyrighted materials. You are allowed to reuse the sample files for only the purpose of theme development for MediaMan. You are not permitted to use these files elsewhere or claim ownership.

Document release information:

Version:	1.0
Release Date:	1/1/2009
Software:	MediaMan 3.0 (build 1025) / MediaMan Library 1.0 (build 1010)*
Author:	He Shiming (heshiming@imediaman.com)
URL:	www.imediaman.com
Change Log:	Version 1.0, initial release.

The following topics are discussed:

1. About MediaMan Theme

Theme is an important part of MediaMan 3. It controls how the items are rendered in virtual shelf view mode, as well as the expanded and large icon view mode. In addition, the item detail pane relies on theme to render HTML content. Functions such as editing, image changing, and auto-suggest of the item detail pane are also implemented by theme through scripting.

By default, MediaMan 3 comes with a theme package file called Essential3.mmp after a successful installation. It's inside C:\Program Files\MediaMan\Data. Upon start, MediaMan will scan this directory for all .mmp files and load them for later use. To change a theme or a package, user will need to press Alt and go to Tools -> Options in MediaMan. Settings will be stored in Windows registry.

* There's no guarantee that the theme you developed now will always work in a future version of MediaMan. It is possible that a future update makes breaking changes that could cause theme packages to be incompatible which eventually led to malfunctioning and crashing. But we'll try to prevent these unfortunates from happening.

This guide is about how to make such theme package files to provide alternative appearances and additional functions to MediaMan. You will need the knowledge and tools illustrated in the following table for development:

Table 1 Required knowledge & tools for development

Required Knowledge	Required Tools & Software
HTML, JavaScript, XML, AJAX, Graphics Design, Webpage Design, Windows Media Player	Notepad++ (or PSPad, CoffeeCup Free HTML Editor), Paint.NET (or Adobe Photoshop), Creative Docs .NET (or Adobe Illustrator), MediaMan Theme Builder

2. Getting Started with Sample

A theme package file is a compressed archive with many files inside. This guide comes with a sample folder that contains these files in their original forms (uncompressed or unpacked). This folder is named "Essential3" and it contains the source files for the theme package that comes with MediaMan 3.

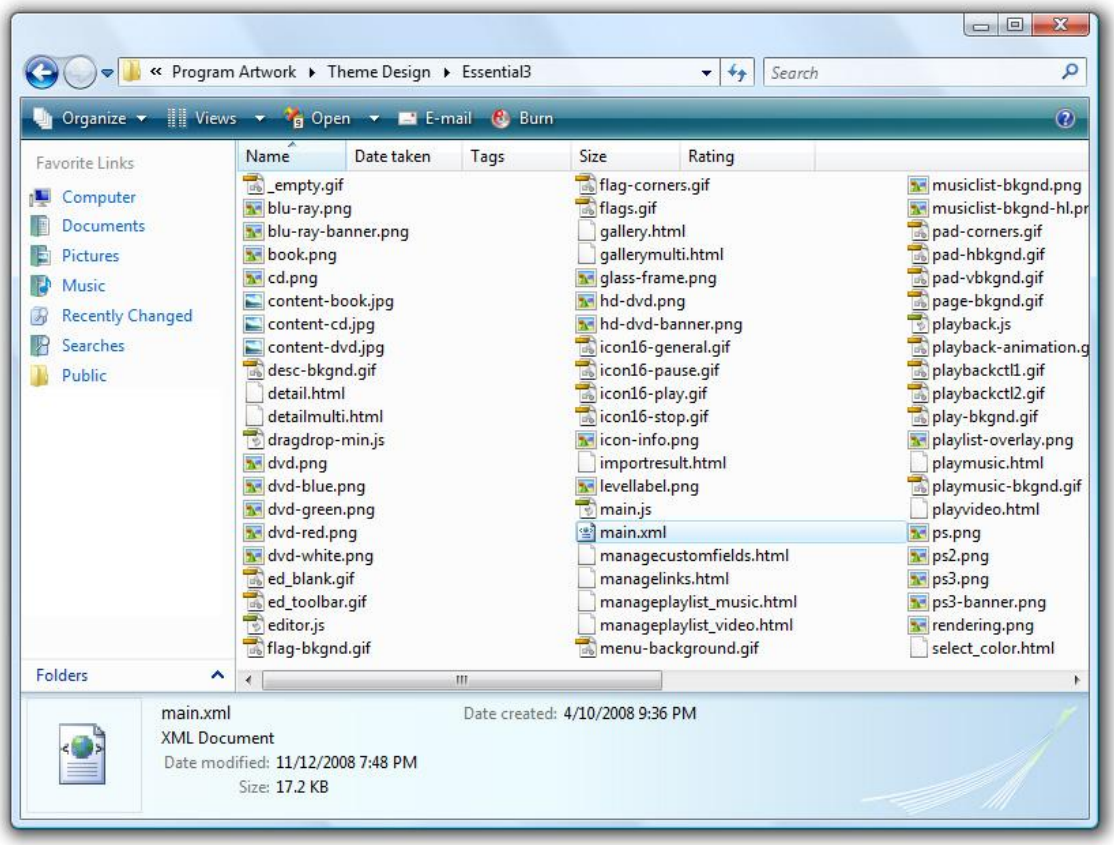

Figure 1 Source files for "Essential3" theme

If you are familiar with HTML and webpage design, you will notice that the content of this folder is no more than a regular website. It's got pictures, webpages, and stylesheets. Nonetheless, the most important file here is **main.xml**, which is the definition file of the theme package, in XML format. MediaMan uses this file to understand what the purpose of

each file is about and more importantly, to decide how to render items in virtual-shelf and the detail pane.

Note that a theme package can only contain a flat list of files. Sub-directories are not supported, and are not recognized by the MediaMan Theme Builder program. Which means you must put all files together inside one working directory.

Essential3's definition file contains lots of nodes and elements. The figure below contains a stripped down version of this definition that only contains the most vital entries:

```
<?xml version="1.0"?>
<mediamanpackage>
  <info>
    <author>He Shiming</author>
    <email>heshiming@imediaman.com</email>
    <url>http://www.imediaman.com/</url>
    <name>MediaMan Essential3</name>
    <description>MediaMan's native theme look for virtual shelf and detail view</description>
  </info>
  <theme name="Color Varation: Red" screenshot="shelf-screenshot1.jpg">
    <templates>
      <detail file="detail.html" mode="detail"/>
      <detail file="detailmulti.html" mode="detailmulti"/>
      <detail file="detail.html" mode="gallery"/>
      <detail file="detailmulti.html" mode="gallerymulti"/>
      <detail file="importresult.html" mode="importresult"/>
      <detail file="playmusic.html" mode="playmusic"/>
      <detail file="playvideo.html" mode="playvideo"/>
    </templates>
    <virtualshelf background="shelf-background1.jpg" uwidth="190"  uheight="230" uhspace="2" uvspace="50">
      <param name="Rendering" image="rendering.png"/>
      <param name="LevelLabel" image="levellabel.png" textcolor="#FFFFFF"/>
      <group name="Book" image="book.png" content="content-book.jpg" olx="9" oly="9" olwidth="172"
olheight="218" autosize="true"/>
      <group name="Music" image="cd.png" content="content-cd.jpg" olx="32" oly="83" olwidth="144"
olheight="144"/>
    </virtualshelf>
  </theme>
</mediamanpackage>
```

Figure 2 Stripped down version of main.xml definition file

It begins with a root node named "mediamanpackage". The "info" node contains descriptive information about the theme package, it's not very important but you still need to be careful about the values to be filled in. All sub-nodes under "info" are required. Pay attention to the "name" node as it's the name of your theme package displayed in the theme selection dialog (the Option dialog).

The "theme" node defines a variation of theme. It has two attributes, "name" and "screenshot". Attribute "name" defines the name of this variation of theme. It's displayed in the theme selection dialog. Attribute "screenshot" contains the file name of a screenshot image of this variation. Consult the sample file for the suggested size of this screenshot. Currently it's not used by MediaMan 3, but it's possible that a future version uses this attribute.

In the real **main.xml** file, you may see multiple "theme" nodes under "mediamanpackage". This means one theme package file can contain multiple variations of themes. Each node of "theme" is used to define a unique set of rendering specifications of virtual-shelf, item detail,

and play music/video. The purpose of multiple variation mechanism is to make it simple for content reusing. So that when you are trying to create several color variations for the same kind of look, you don't have to create multiple theme packages.

It's important to ensure your **main.xml** is well-formed and contain valid information. Otherwise, MediaMan may simple crash upon start without producing any helpful message.

3. Virtual-Shelf Rendering Specifications

Node "virtualshelf" under "theme" defines how MediaMan renders items in virtual-shelf view mode. When virtual-shelf view mode is selected (installation default), MediaMan takes item cover pictures, and combine it with some pre-defined pictures to compose a realistic rendering of a book, or CD.

```
<?xml version="1.0"?>
<mediamanpackage>
  <theme name="Color Varation: Red" screenshot="shelf-screenshot1.jpg">
    <virtualshelf background="shelf-background1.jpg" uwidth="190"  uheight="230" uhspace="2" uvspace="50">
      <param name="Rendering" image="rendering.png"/>
      <param name="LevelLabel" image="levellabel.png" textcolor="#FFFFFF"/>
      <group name="Book" image="book.png" content="content-book.jpg" olx="9" oly="9" olwidth="172"
olheight="218" autosize="true"/>
      <group name="Music" image="cd.png" content="content-cd.jpg" olx="32" oly="83" olwidth="144"
olheight="144"/>
    </virtualshelf>
  </theme>
</mediamanpackage>
```

Figure 3 Stripped down version of main.xml focusing on virtual shelf

Node "virtualshelf" itself has 5 attributes, all are required. The "background" node contains the file name for the background picture (the shelf, open the sample files to see). Attributes "uwidth" and "uheight" refer to "unit width" and "unit height". They are the pixel dimension of the size of a single item on shelf. The group images (to be discussed later) must match this size exactly. Attributes "uhspace" and "uvspace" refer to "unit horizontal space" and "unit vertical space". They define the gap between items in pixels. Only one "virtualshelf" node can be defined under "theme". To define an alternative background appearance or size, you have to create another variation by defining another "theme" node.

There are two "param" nodes under "virtualshelf" and both must be exist. They are used to define some essential parameters. Attribute "name" of the first one must be "Rendering". The "image" attribute of it points to an image file* used as a place-holder when the item is not rendered. This image must match the unit width and height specified in the "virtualshelf" node. The second "param" node defines "LevelLabel".

* It's a 32-bit PNG image with an 8-bit alpha channel. Alpha channel made it possible to blend (compose) two images together when either or both of them are semi-transparent (translucent). MediaMan uses 32-bit PNG images a lot to render blended images. Most images in a theme are required to be 32-bit PNG. Visit http://en.wikipedia.org/wiki/Alpha_compositing if you are new to this concept.

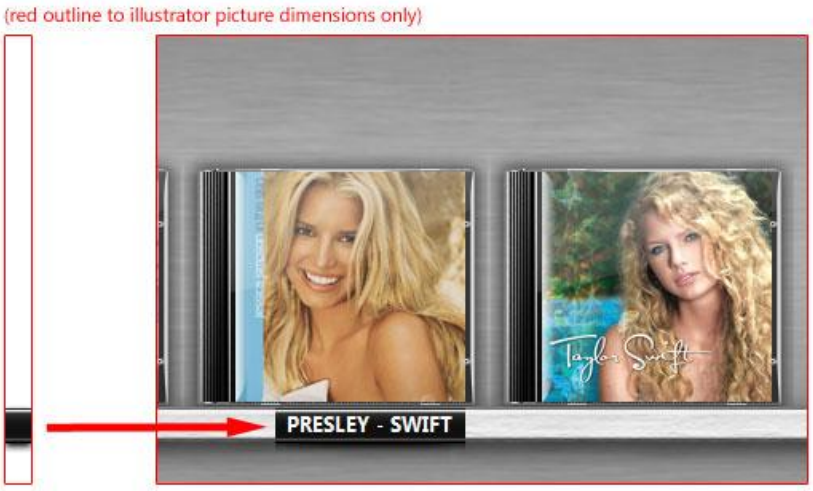

Figure 4 "LevelLabel" in practice

The figure above illustrates how MediaMan implements a "LevelLabel". It's an image whose height matches the unit height. A "textcolor" attribute must be specified in HTML style color syntax. MediaMan take this "LevelLabel" image, expand it horizontally just enough to fill with text. This image is then blended with the background shelf image. So the actual position of this label depends on the background image. In real situations, this label image doesn't have to be narrower than the shelf. But if you intend to put textures on it, keep in mind that it'll be sized up or down, and it won't be repeated.*

The color of the label text must be specified. However, as of the current design, it's not possible to specify the position of the text. It's hard coded in MediaMan that the label height is always 1/15 of the background height, and label vertical position is always 9/10 starting from the top of the background image. This design is likely to change in the future to allow further flexibility of the label design.

Group images define how MediaMan render each and every item in virtual-shelf. They are defined through the "group" node. In Figure 3, two "group" nodes are provided. While in the real main.xml definition, dozens of group images are provided to allow customizations. The difference of these two "group" nodes is that the first node for "Book" contains an extra attribute named "autosize", and it's set to "true".

We'll first talk about the second "group" node as it's easier to explain.

Internally, MediaMan uses the field "Group" to identify the nature of an item†. This field is also called "Packaging" sometimes in the user interface to provide a better understanding. The reason for this is that Amazon returns consistent information for this field for all items. Values "Music", "Book" are two examples of what Amazon returns for certain items. For a list

* The "LevelLabel" mechanism provided a way to quickly identify items by showing portions of their titles. In MediaMan, this rendering can be turned off from the option dialog. Regardless of this fact, the "LevelLabel" parameter must exist in the theme.

† MediaMan doesn't attempt to recognize an item by "Medium" or "Media Type". Nor does it check for the nature of items by fields like "ISBN" or "Theatrical Release Date". The rendering depends solely on this "Group" field.

of possible values used by Amazon as well as MediaMan, please refer to Appendix II. Recommended Product Groups/Packaging on page 17.

Figure 5 below shows how MediaMan uses group image file "cd.png" (a container image) together with attribute "olx", "oly", "olwidth", and "olheight" to compose a rendering of an item.

Figure 5 Group image composition procedures for "Music"

Attribute "olwidth" and "olheight" stands for "overlay width" and "overlay height". These two attributes determine the target size of the image inside the container. And "olx", "oly" which stands for "overlay x position", "overlay y position", determine where the image should be located in the container image.

In real world, a CD cover is always square and a DVD cover is always a rectangle with a fixed aspect ratio. If you stack a couple of these items together, you'll discover that they are in the exact same size regardless the types of music or the genre of the movie. That's why we can make a simple empty container, resize the picture, and render a pseudo look for such items. The virtual-shelf mechanism utilizes this method to render all items.

Last but not least, in case the user didn't import a cover image, MediaMan will attempt to provide a default rendering. Attribute "content" is used to specify a background image to occupy the target overlay area. There isn't a strict limitation on the size or ratio of this image,

nor is its color especially important. MediaMan will resize it to fit the specified area, and decide a text color based on the brightness of the content image.

Now let's take a look at the "group" node for "Book". As is mentioned earlier it's a bit different because the extra "autosize=true" attribute. The composition procedure for this node is actually the same as Figure 5. However, please note that the group image and the parameters define the largest form of this type of items. The actual rendering of the items will be as large as what you've specified, or narrower in width, or shorter in height.

This mechanism allows items such as books to be rendered in difference sizes and keeping their original shape and aspect ratio.

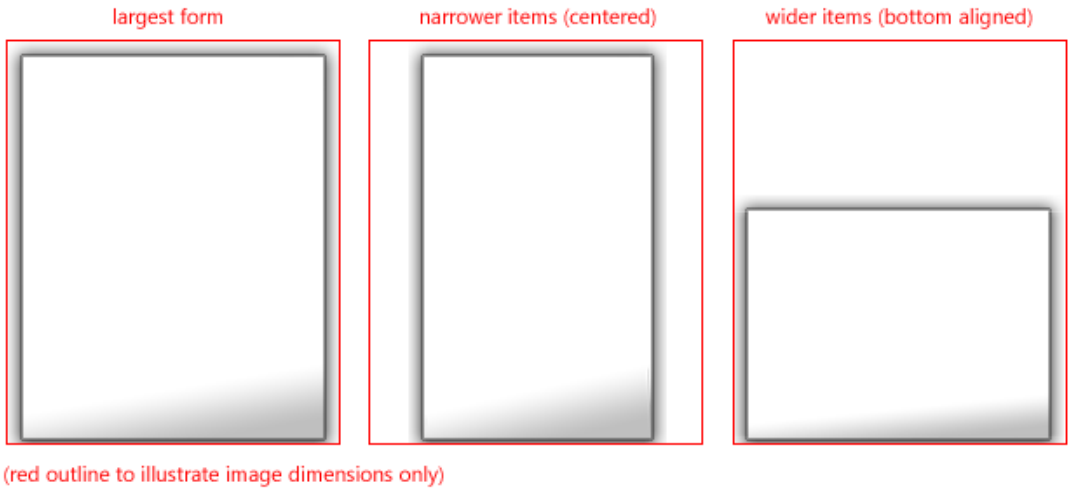

Figure 6 Rendering "autosize=true" group images

MediaMan only resizes the overlay area related borders to fit the target image aspect ratio. It doesn't resize the entire image. This means the corners and borders will be kept intact during resizing, so that the group image won't get distorted when the target image is too narrow or wide.

It's currently not possible to specify the alignment of these auto-sized renderings. It's hard coded that the image is always centered and bottom aligned.

Currently, the "Essential3" theme comes with MediaMan specifies a 190 x 230 unit bounding box. It's not square. At the same time, regardless of the actual item size, MediaMan must render all items in the same size aligned in a fixed grid. Therefore, packaging such as DVD and books tend to appear tightly placed and larger. On the other hand, CDs and wider books appear smaller and loosely placed. Additionally, the rendering might produce a false visual about the size. Future versions of MediaMan will try to overcome these limitations. In the mean time, it's still possible to overcome these shortcomings by introducing extra groups into the theme, such as "Book (Small)", "Book (Medium)".

Also note that although a theme package is composed with files, MediaMan doesn't actually decompress them and put them into a directory of some kind. The decompression procedure is done when MediaMan starts and decompressed files are put in the memory. It's

not possible to access them directly once they are converted to a theme package file. Access can be only granted via the means MediaMan provides.

4. Detail HTML Rendering Specifications

MediaMan renders item details in HTML*. With the help of a hosted Internet Explorer instance, MediaMan is able to display this rendered HTML directly within the program window.

It works very similar to an HTTP protocol based website. When the user click an item, MediaMan issues a navigate command to the hosted Internet Explorer instance with a particular URL syntax recognizable internally by the program. Internet Explorer then requests the server inside MediaMan to serve this URL. The server renders the content and then sends output to Internet Explorer.

It is also possible for Internet Explorer to retrieve a particular file from the theme package or MediaManRes.dll so that files such as .GIF, .PNG, .CSS, and .JS can be served to customize styles and designs. However, as is noted before, these files reside in memory only. MediaMan won't unpack them into a particular directory.

Technically, MediaMan uses Asynchronous Pluggable Protocol (APP)† to achieve this feature. The URL syntax begins with "mediaman://". As of the current version, it actually begins with "mediaman://mediamandataserver/". As far as zone security is concerned, MediaMan maps this URL to HTTP‡, so that it shares the "Internet Zone" security settings with Internet Explorer.

Let's take a look at the HTML rendering part definition:

```
<?xml version="1.0"?>
<mediamanpackage>
  <theme name="Color Varation: Red" screenshot="shelf-screenshot1.jpg">
    <templates>
      <detail file="detail.html" mode="detail"/>
      <detail file="detailmulti.html" mode="detailmulti"/>
      <detail file="detail.html" mode="gallery"/>
      <detail file="detailmulti.html" mode="gallerymulti"/>
      <detail file="importresult.html" mode="importresult"/>
      <detail file="playmusic.html" mode="playmusic"/>
      <detail file="playvideo.html" mode="playvideo"/>
    </templates>
  </theme>
</mediamanpackage>
```

Figure 7 HTML rendering definition in main.xml

¨ All HTML files must specify UTF-8 character set in the <head> tag. Regardless of the actual encoding, MediaMan will treat it as UTF-8 encoding and perform Unicode conversion before processing the content.

† For a description of APP, visit http://msdn.microsoft.com/en-us/library/aa767743(VS.85).aspx .

‡ It means "mediaman://mediamandataserver/" is equivalent to "http://mediamandataserver/" in terms of Internet security. Settings such as cookies, scripting, ActiveX execution of the Internet Zone also applies in MediaMan. As of Internet Explorer 7, this mechanism made it impossible to access files on your hard drive directly. This mechanism also made it possible to refer the content on a website on the Internet from within the item detail pane of MediaMan. However, you should not rely on this mechanism to design a theme. A future version may introduce extra security measures to forbid referencing content from an Internet URL, or restrict referencing.

Node “templates” appears under “theme”. All sub-nodes under it are named “detail”. Attribute “file” points to an HTML file, and “mode” defines how MediaMan uses this file. There are 7 modes and they must all be present. “playmusic”, “playvideo”, and “importresult” mode are to be discussed in separated chapters. “detail” template is used when the user selected one item, “detailmulti” template is used when the user selected more than one items. “gallery” and “gallerymulti” must be present for historical reasons. They are no longer used in MediaMan 3 because the “detail” template is now used for the gallery (Image only item detail).

If you open up the sample **detail.html** or **detailmulti.html**, you will notice that they are regular HTML files but contain rather complicated functions. They link to a **main.js** file that provides features such as editing. Let’s take a look at a stripped down version of **detail.html**:

```
<title>[FieldValue:0x10000006]</title>
<script type="text/javascript">
        var _dateFormat = "[SpecialInfo:DateFormat]";
        var _detailMode = "[SpecialInfo:DetailMode]";
</script>
...
<form id="form_update" action="mediaman://mediamandataserver/update/" method="post" ...>
<input type="hidden" name="x10000002_hidden" value="[FieldValue:0x10000002]" />
<input type="hidden" name="x1000000c_hidden" value="[FieldValue:0x1000000c]" />
<input type="hidden" name="x10000038_hidden" value="[FieldValue:0x10000038]" />
...
<tr id="x1000003f_container">
        <td id="x1000003f_fieldname" class="fieldname">[FieldName:0x1000003f]</td>
        <td id="x1000003f_subcontainer"><span id="x1000003f">[FieldValue:0x1000003f]</span></td>
</tr>
<tr id="x10000038_container">
        <td id="x10000038_fieldname" class="fieldname">[FieldName:0x10000038]</td>
        <td id="x10000038_subcontainer"><span id="x10000038">[FieldValue:0x10000038]</span></td>
</tr>
<tr id="x1000000c_container">
        <td id="x1000000c_fieldname" class="fieldname">[FieldName:0x1000000c]</td>
        <td id="x1000000c_subcontainer"><span id="x1000000c">[FieldValue:0x1000000c]</span></td>
</tr>
```

Figure 8 Stripped down version of detail.html

Figure 8, there is “[FieldName:0x1000000c]” and “[FieldValue:0x1000000c]”. This is a **template tag** internally recognizable by MediaMan*. The HTML rendering routines will look for these predefined template tags and replace them with actual content retrieved from the collection. If you take a quick look at Appendix I. Field IDs on page 16, you’ll see that field “0x1000000c” is the Release Date field. So when MediaMan renders this page, it’ll replace “[FieldValue:0x1000000c]” with the actual release date specified for the item (for instance, 12/13/1989). “[FieldName:0x1000000c]” will be replaced with text “Release Date”. Start MediaMan and navigate to any records to see this conversion, and to compare rendered text with the template.

“0x1000000c” is actually a hexadecimal number used by MediaMan to identify a field. Appendix I. Field IDs on page 16 provides a full list of all the number IDs MediaMan uses for fields.

* Note that this is a fixed syntax style with square brackets, text, semicolon, and a hexadecimal number. If you write it correctly, it won’t conflict with any HTML or XHTML standards.

Likewise, "[FieldValue:0x10000006]" in the <title> tag, will replaced by the title of the item. However, since "[FieldName:0x10000006]" isn't found in **detail.html**, the term "Title" won't appear anywhere. All fields are opt-in.

When you are starting out from scratch, it is recommended that you start a blank web page, and put only the "FieldValue" and "FieldName" template tags to understand how it works.

HTML rendering is very simplified in MediaMan. The program itself is responsible only for replacing those recognized template tags only. It doesn't parse the HTML file as a document object. Nor does it produce any HTML tags. In addition, the replaced values come in their raw form, almost exactly like what's stored in the collection file. If the original value is empty, the replaced value will be just empty. And for multi-value fields, such as the Artist field, values are separated by a CRLF return character, rather than
 let alone .

Since the rendering routine does not process these situations, we have to rely on Javascript to process them after we saw them (i.e. the page is loaded). This is typically done with a handler function specified in "onload" attribute of the body tag. The actual implementation can be various.

In Figure 8, you see a <tr> tag wrapped around the value for release date. It has "x1000000c_container" specified in the "id" attribute. Additional IDs such as "x1000000c_fieldname", "x1000000c_subcontainer", "x1000000c" can also be found. Plus, there is a big <form> element covers the entire page. It has got lots of hidden input fields such as "x1000000c_hidden". They are the basis to provide a better rendering of the raw content, through JavaScript.

In a nutshell, these IDs and names are used this way*:

1. MediaMan replaces "[FieldValue:0x10000006]" with the content
2. when the page is loaded, a JavaScript function is triggered
3. this function checks whether the value of the hidden input field "x1000000c_hidden" is empty, if it is, it locates the <tr> tag by ID "x1000000c_container" and set its style.display to "none", so that it's hidden
4. if this is a multi-value field like the artist field, it takes the hidden input field value, replace the CRLF ("\n") with "
", or form an unordered list using <ul> and <li>, then replace the innerHTML of "x1000000c" to this newly generated content.

The other two ID attributes not mentioned are to be used by the Editing function. Note that although these attributes are named in similar ways as the FieldValue, FieldName template tags, they are not to be recognized and parsed by MediaMan's HTML rendering routines.

* For the description field, check "process_ExpandDescription" function in **main.js** to see how it expands the HTML content. The tracks field uses a double CRLF (carriage return, or "\n") to separate discs, and a single CRLF to separate tracks. See "process_ExpandTracks" to understand how it works. The customer review content is split to 3 fields, FIELD_CRSUMMARY, FIELD_CRCOMMENT, and FIELD_CRRATING. Inside these fields, content is further delimited by a single CRLF. The content of FIELD_CUSTOMFIELDS is delimited by a single CRLF, then a single TAB ("\t") to separate field name and field value.

Another aspect of the **detail.html** template is the image. MediaMan provides simply ways to retrieve the original cover images, as well as rendered virtual-shelf style images.

To retrieve a virtual-shelf rendering PNG image, simply insert an image tag, and point its "src" attribute to a URL like this "mediaman://mediamandataserver/retrieve/?output=png&style=virtualshelfcrop&records=id[6]&recordtype=item&size=pixel[100]"*. There are two parameters you can customize with this URL. The first one is the "records=id[#]" part. It uses the unique identifier (unique ID) of the items to determine which one to render. Its corresponding field ID is 0x10000002, which means you can get this value by putting "[FieldValue:0x10000002]" anywhere on the HTML page. Unique ID is a base 10 number. The number "100" in "size=pixel[#]" defines the maximum width of the rendered output. Parameter "style" can either be set to "virtualshelf" or "virtualshelfcrop". When rendering in the "virtualshelfcrop" style, MediaMan will crop all the complete transparent pixels.

To retrieve the original cover image, use a URL like "mediaman://mediamandataserver/retrieve/?output=jpg&style=currenttheme&records=id[6]&range=attachment[0]&recordtype=item". The "range=attachment[#]" part of this URL uses a field called FIELD_NUMATTACHMENTS ([FieldValue:0x11000001]). An "attachment" is a phrase internally used by MediaMan to refer to an image associated with an item. An item can contain multiple attachments, in which case, the value of the FIELD_NUMATTACHMENTS will be rendered as "Front Cover:0,Disc:1" etc (refer to the notes section in Appendix I. Field IDs on page 16 for more information). To implement cover viewing capabilities, you can put a hidden input field to retrieve this value, and then use JavaScript to split it up and produce such image tags. Again, check **detail.html** for an example of this feature.

There are other template tags designed to make program parameters available for use with JavaScript. In Figure 8, two tags are found in the <script> tag in <head>. They are [SpecialInfo:DateFormat] and [SpecialInfo:DetailMode]. The "DateFormat" tag will be replaced by a date format that comes in the format of "M/d/yyyy" or "yyyy/M/d" depending on the locale settings on the user's computer. This is a clue for JavaScript functions to learn about the format of the date fields.

[SpecialInfo:DetailMode] will be replaced by either "" (empty),"edit", "gallery", or "galleryedit". This value reflects the state of the "Edit" button on toolbar, and the state of the toolbar below the HTML item detail pane (either "Detail" is pressed or "Image" is pressed). Use this value to decide what to display, or prepare for editing.

Also note that when the user click "Edit", or "Image", MediaMan will not attempt to refresh the item detail page. Therefore in addition to handling the previous special value "[SpecialInfo:DetailMode]", you also needed to provide 2 JavaScript functions for MediaMan, for the event of clicking these two buttons. The prototypes of these two functions are:

```
function _SwitchDetail(bEdit){}
function _SwitchGallery(bEdit){}
```

* With MediaMan running, you can start Internet Explorer and enter this URL to see the result just like in MediaMan. Make sure you selected an appropriate unique ID.

The parameter "bEdit" will be set to either true or false depending on whether it's entering or leaving the edit mode. When the user clicks "Image" on the toolbar below the HTML detail pane, _SwitchGallery will be called. Likewise, _SwitchDetail will be called when "Detail" is clicked. The syntax of these two functions is hard coded into MediaMan and can't be customized. Although a future version may change this syntax.

In the real **detail.html**, these two functions are merely there to hide certain parts of the HTML page.

5. Rendering Multiple Items

As is mentioned before, MediaMan uses an alternative template when multiple items are selected*. The rendering procedures are exactly the same. However, FIELD_ID ([FieldValue:0x10000002]) will contain all the FIELD_IDs for the selected records delimited by a comma. For all fields, MediaMan will put either "Something", or "Something (3 different values)" as the "FieldValue" depending on the actual situation. If no "(# different values)" are found in the text, it means this field is the same for all selected items.

The sample implementation **detailmulti.html** provided very limited information to display. Elements such as virtual-shelf rendering, and customer reviews are not displayed. This is not a limitation; it is possible to display all of them. When designing the content, make sure they are valuable to the user. Sometimes presenting everything isn't helpful. With multiple items selected, the user is more likely to perform a batch edit operation. The sample implementation will dim the field when it contains different values, so that the user can focus on those fields containing the same value.

6. Implementing "Import Result"

"Import Result" of the Import from Web feature in MediaMan can also be customized via theme. When a search query is completed, MediaMan will attempt to use the HTML item detail pane to display the search result. During searching, MediaMan also fetches images. So it's possible to render the virtual-shelf icon in this search result. "Import Result" works like rendering multiple items. However, instead of rendering "3 different values", it's capable of rendering the details of all items. To implement this feature, a repeat block is introduced.

Please take a look at the sample file **importresult.html** to understand how it works. It's similar to rendering items, although there isn't that much to customize. Note these HTML comment tags:

```
<!--Define:ItemBlockBegin-->
…
<!--Define:ItemBlockEnd-->
```

* Currently there is a limitation of the theme design that MediaMan cannot display pictures when multiple items are selected. This is rather a flaw but not a technical limitation in the current design, and is very likely to be improved in the near future. At that time, the current implementation of cover viewing feature might break.

The content between these tags will be used to render a single item in the result. In other words, the <table> tag works like a repeat block, and it will be repeated for each item in the result. The ID of the table is set to "result-item-[SpecialField:ItemIndex]". There is a special template tag here, and it will be replaced to the index of the item in the result. For each item, a special index will be assigned. The tag "[SpecialField:0x11001001]" will be replaced to the virtual-shelf icon.

The "onclick" event of this table points to a function called "processImportSelection". It's located in **main.js**. And the purpose of it is to tell MediaMan to select a record and finish the import. This function will check for Ctrl key states and call "window.external.ImportResult_Select(nIndex, bKeepList);" to issue the command to MediaMan. The first argument of this function is the index discussed in the previous paragraph. The second is a Boolean value to tell MediaMan if it should keep other items in the result.

There is another function named "Init_ImportResult();" in **main.js** to establish a checkbox filtering function. It's implemented by iterating all the <table> tags and figure out items types by reading out the value of FIELD_GROUP.

It is recommended that you don't make big changes to the "Import Result" template and keep it as is.

7. Implementing Editing

Editing is implemented via JavaScript and XML posting over HTTP in MediaMan 3. It works the same way as so called "Web 2.0 applications" or "AJAX applications". Template **detail.html** illustrated in previous chapters is responsible for gathering input text, compile an XML document describing the new item information, and submit to a particular URL. The server that handles this submission will report back whether the change was successful.

The routines are in the **main.js** of the sample files, although the source code is not written to be illustrative and instructive.

Given that you finished the steps creating the rendering template, you can follow these steps to implement the editing function:

1. When the edit command is initiated, install "onclick" handlers to field names and field values. Given that they all have their unique ID attribute in HTML, this can be done in a batch. Alternatively, you can put "onclick" handler directly in the HTML tag, and have the handler check if we are in the "Edit" mode.
2. To implement editing in this handler, a typical way would be to replace the field value content with an edit box. JavaScript and HTML DOM are very helpful in the implementation. You can take the value in the hidden input field; decorate it with a text input field <input>, or a text box field <textarea>. Then, replace the value with this input box by using the innerHTML attribute of that tag.
3. Make sure that you put the keyboard focus into this newly created input box by using the focus() method of this object. So that when the focus left, we can confirm this change. This is done by putting an "onblur" handler in the input object. This handler should fetch the value from the input box, and put it back to the hidden input field,

then replace the input box back to the newly entered value by using the innerHTML attribute[*].

4. There are other more complicated fields, such as a multi-value field like artist, a date field, an HTML-capable description field, and the track listing field which supports multiple tracks on multiple discs. These fields should be taken care of by their special routines[†].
5. Keep an internal list of all modified fields. We'll submit only these fields to MediaMan for a quick update.[‡]
6. When the user finishes the editing, he'll click the "Save" button on top of the detail pane to commit. This button is customizable through **detail.html**. When this button is triggered, the script is responsible to compose an XML document containing the changes and submit it to "mediaman://mediamandataserver/update/?input=xml&recordtype=item&action=partialupdate" via POST method.[§]
7. The script then checks the return value and prompt user if the submission is failed.
8. To implement "Import Image", call "window.external.ImportImage(0);".
9. To implement "Replace Image", call "window.external.ReplaceImage(0,#TEXT,#ID);", where both #TEXT and #ID is retrieved via FIELD_NUMATTACHMENTS, for example, "window.external.ReplaceImage(0, "Front Cover", 0);".
10. To implement "Remove Image", call "window.external.RemoveImage(0,#ID);", where #ID is the same attachment ID in the previous step.

This article doesn't talk about the sample implementation of auto-suggest. It uses persistence[**] to store the entered values. Which means the implementation is completely subject to the template page. It has nothing to do with the MediaMan program itself.

If you are unfamiliar with this, you can customize the style by using pure CSS without changing any of these functions. If you encounter a problem, make sure to compare with the sample file for a clue of the difference.

8. Implementing Play Music & Play Video

Play Music / Video currently rely on theme; however, it's possible that a future update may no longer use themes to accomplish these functions. Although as of MediaMan 3.0 build 1025, Play Music / Video is still a must have feature for a theme package.

---

[*] See "handler_TextFieldEnterEdit" in **main.js** for a sample implementation.

[†] This is mentioned in an early chapter on Detail HTML rendering. See "handler_TextAreaEnterEdit", "handler_TracksEnterEdit", "handler_HtmlEnterEdit", "Manage_CustomFields" in **main.js**, and **managecustomfields.html** to learn about converting these fields to editable input boxese.

[‡] Search for "_ModifiedFields" in **main.js** to see how the sample implementation maintains this list.

[§] This XML document is based on a template of "<mediamanlw><items><item></item></items></mediamanlw>". Inside the "item" node, you need to have fields listed like "<field id="x10000006">Content</field>". See "Commit_Changes", "Commit_Changes_XMLHelperInsertNode" in **main.js** for a sample implementation.

[**] Persistence is a feature of Internet Explorer. Persistence made it possible to store some simple data at user's local computer with client side scripting. Please refer to http://msdn.microsoft.com/en-us/library/ms531066(VS.85).aspx if you are new to this subject.

```
<?xml version="1.0"?>
<mediamanpackage>
  <theme name="Color Varation: Red" screenshot="shelf-screenshot1.jpg">
    <templates>
      <detail file="detail.html" mode="detail"/>
      <detail file="detailmulti.html" mode="detailmulti"/>
      <detail file="detail.html" mode="gallery"/>
      <detail file="detailmulti.html" mode="gallerymulti"/>
      <detail file="importresult.html" mode="importresult"/>
      <detail file="playmusic.html" mode="playmusic"/>
      <detail file="playvideo.html" mode="playvideo"/>
    </templates>
  </theme>
</mediamanpackage>
```

Figure 9 "Play Music/Video" part in the theme definition

Its rendering is actually the same as rendering item details when multiple items are selected. In order to render this page, MediaMan simply select all items that contain something in the "File Listing". Regardless of what file types are actually linked, MediaMan will output all of them in FIELD_FILELIST. The template is responsible of deciding what files are playable and how to construct a list. When rendered in HTML, the content of FIELD_FILELIST is delimited by two CRLF returns. For the file list of each record, the first line is a double-quoted directory path, and the rest are double-quoted file names. This made it possible to dynamically construct a play list for the Windows Media Player ActiveX control* to playback these files.

Please note that MediaMan will automatically elevate the security zone from "Internet" to "Local Computer" when navigate to Play Music or Play Video. This means the Windows Media Player ActiveX object will have full access to these files include features like obtaining ID3 tags.

9. Using MediaMan Theme Builder

MediaMan Theme Builder is a tool to pack up your theme folder, and produce a "MediaMan Package File" or ".mmp" for MediaMan†. To use it, simply click "Browse" and pick the folder that contains theme files, and click "Build"‡. Observe the message box and check if anything unusual is found within your theme package.

Once the file is built, it'll put to the parent directory of your theme folder. Copy this file to C:\Program Files\MediaMan\Data . Restart MediaMan, and select this theme using Alt, Tools -> Options and see if it works correctly. Please note that the theme builder currently doesn't check the correctness of the theme definition. If MediaMan crashed upon start or when you select a theme, it means there's a problem with the definition. Image dimension issues, HTML encoding issues, and JavaScript issues are very unlikely to cause a crash in MediaMan. Therefore, if anything goes wrong, make sure you first check **main.xml** and compare it with the sample file.

---

* For more information about Windows Media Player ActiveX control, visit http://msdn.microsoft.com/en-us/library/bb249259(VS.85).aspx .

† MediaMan Theme Builder is not compatible with User Account Control (UAC) of Windows Vista. Please don't put it in C:\Program Files\ or any other directories protected by the UAC mechanism.

‡ The theme builder current allows you to build up to 5 folders into 5 theme packages at the same time, use the first "Browse" button if you only have one to build.

Appendix I. Field IDs

FIELD_ID	0x10000002
FIELD_SUBID	0x10000003
FIELD_AMAZONSITE	0x10000004
FIELD_ICON	0x10000005
FIELD_TITLE	0x10000006
FIELD_STATUS	0x10000007
FIELD_ASIN	0x10000008
FIELD_NUMMEDIA	0x10000009
FIELD_MEDIA	0x1000000a
FIELD_PUBLISHER	0x1000000b
FIELD_RELEASEDATE	0x1000000c
FIELD_GROUP	0x1000000d
FIELD_DETAILURL	0x1000000e
FIELD_UPC	0x1000000f
FIELD_EAN	0x10000010
FIELD_IMAGEURLLARGE	0x10000011
FIELD_IMAGEURLSMALL	0x10000012
FIELD_IMAGEURLMEDIUM	0x10000013
FIELD_DESCRIPTION	0x10000014
FIELD_ASPECTRATIO	0x10000015
FIELD_DVDLAYERS	0x10000016
FIELD_DVDSIDES	0x10000017
FIELD_PICTUREFORMAT	0x10000018
FIELD_REGION	0x10000019
FIELD_RUNNINGTIME	0x1000001a
FIELD_STUDIO	0x1000001b
FIELD_THEATRICALRELEASE	0x1000001c
FIELD_AUDIENCERATING	0x1000001d
FIELD_ACTORS	0x1000001e
FIELD_DIRECTORS	0x1000001f
FIELD_FORMAT	0x10000020
FIELD_LANGUAGE	0x10000021
FIELD_PUBLICATIONDATE	0x10000022
FIELD_NUMPAGES	0x10000023
FIELD_ORIGINALTITLE	0x10000024
FIELD_AUTHORS	0x10000025
FIELD_TRANSLATORS	0x10000026
FIELD_ISBN	0x10000027
FIELD_LABEL	0x10000028
FIELD_NUMTRACKS	0x10000029
FIELD_ARTISTS	0x1000002a
FIELD_TRACKS	0x1000002b
FIELD_ESRBRATING	0x1000002c
FIELD_FEATURES	0x1000002d
FIELD_PLATFORMS	0x1000002e
FIELD_AVGCUSTOMERRATING	0x1000002f
FIELD_TOTALCUSTOMERREVIEW	0x10000030
FIELD_CRRATING	0x10000031
FIELD_CRSUMMARY	0x10000032
FIELD_CRCOMMENT	0x10000033
FIELD_CUSTOMFIELDS	0x10000034
FIELD_AMAZONOFFERLSTID	0x10000035
FIELD_AMAZONPRICE	0x10000036
FIELD_AMAZONAVAIL	0x10000037
FIELD_DATEADDED	0x10000038
FIELD_LISTPRICE	0x10000039
FIELD_LOWESTNEWPRICE	0x1000003a
FIELD_TOTALNEW	0x1000003b
FIELD_LOWESTUSEDPRICE	0x1000003c
FIELD_TOTALUSED	0x1000003d
FIELD_NOTES	0x1000003e
FIELD_RATING	0x1000003f
FIELD_SUBJECTS	0x10000040
FIELD_GENRE	0x10000041
FIELD_AUDIOFORMAT	0x10000042
FIELD_CATALOGNO	0x10000043
FIELD_CUSTOMSTATUS	0x10000044
FIELD_FILELIST	0x10000045
FIELD_STATUSTARGET	0x10000046
FIELD_STATUSBEGINDATE	0x10000047
FIELD_STATUSENDDATE	0x10000048
FIELD_NUMCOPIES	0x10000049
FIELD_LOCATION	0x1000004a
FIELD_TAG	0x1000004b
FIELD_SERIES	0x1000004c
FIELD_VOLUME	0x1000004d
FIELD_AWARDS	0x1000004e
FIELD_SALESRANK	0x1000004f
FIELD_MPN	0x10000050
FIELD_CREATOR	0x10000051
FIELD_PACKAGEDIMENSIONS	0x10000052
FIELD_DEWEYDECIMALNUMBER	0x10000053
FIELD_IMDBRATING	0x10000054
FIELD_FILMLOCATION	0x10000055
FIELD_PLOT	0x10000056
FIELD_LIBTOTALCOPIES	0x10000057
FIELD_LIBCOPIESLEFT	0x10000058
FIELD_NUMATTACHMENTS	0x11000001
FIELD_ATTACHMENTIDS	0x11000002
FIELD_ATTACHMENTTYPES	0x11000003
FIELD_ATTACHMENTSIZES	0x11000004
FIELD_ATTACHMENTCHECKSUMS	0x11000005

Notes: FIELD_SUBID defines the category unique identifier number an item is assigned to. It might not be suitable to render the content or use it for editing. Fields FIELD_STATUS, FIELD_STATUSTARGET, FIELD_STATUSBEGINDATE, FIELD_STATUSENDDATE are used to define flags. Though all of them are listed, only FIELD_STATUS should be used.

Fields such as FIELD_NUMATTACHMENTS refers to the number of images associated with an item. Although when rendering the HTML, only FIELD_NUMATTACHMENTS will be rendered. Its format will be "ImageName:#ID,ImageName:#ID" (#ID is the attachment ID, not the item ID), therefore the actual render would be "Front Cover:0,Disc:1" etc. A theme should never assume the existence of attachments.

Certain fields, such as FIELD_ICON are defined but is no longer, or never used. Inside a collection file, most fields contain text data, despite of the fact that numeral data is stored. Consult the sample **detail.html** and compare it with the rendering output of MediaMan to understand how each field works.

All of the fields mentioned in this note should not be manipulated using theme's Editing function directly.

Appendix II. Recommended Product Groups/Packaging

The recommended group names are:

- Book
- DVD
- DVD-Red
- DVD-Green
- DVD-Blue
- DVD-White
- HD DVD
- HD DVD (Banner)
- Blu-ray
- Blu-ray (Banner)
- Music
- PS
- PS2
- PS3
- PS3 (Banner)
- Wii
- Wii (Banner)
- XBOX
- XBOX360
- Video Games
- Software
- Glass Frame

This is actually a list of groups (package looks) implemented in the sample Essential3 theme. It is recommended to implement all of them for compatibility, though you are free to come up with your own names for groups. It's not necessary to use different pictures for each of them. If MediaMan encounters a "product group" that is not defined in your theme, it'll fall back to the first one defined. In the case of Essential3, it's "Book".

When you are creating your own "Product Groups", make sure they are similar to ones in this list. Use a caption style capital letter combination. This will ensure the compatibility between third-party themes and make it easier for the user to switch.

Appendix III. Contributing and Proposing Changes

You can contribute and propose new ideas to extend or fix the theme mechanism. Simply visit www.imediaman.com/smf for community discussion, or contact heshiming@imediaman.com directly.