Advanced Interactivity Development Best Practices

Updated: February 21, 2008

Amy Dullard
Program Manager, HD DVD

Revision History

Date	Author	Description
12/11/2007	Amy Dullard	Document initiated.
02/21/2008	Amy Dullard	Typos fixed, clarifications added

Table of Contents

Introduction

This document is a set of guidelines for developers of advanced content for HD DVD. Following the best practices in this document should help developers create applications that are compatible with and function well on all HD DVD players running HDi™.

About HDi™

HDi™ is Microsoft's implementation of the advanced interactivity stack for HD DVD. Toshiba HD DVD players and the Xbox 360 HD DVD add-on run HDi™. Content providers may add the HDi™ logo to their packaging provided they follow the guidelines set forth in this document and pass the content certification process approved by Microsoft.

Pre-Development

Specifications

Functional Specification

Before design or development begins, a functional specification should be drafted detailing the intended use and goals of the application being developed.

A functional specification should go into detail on interaction design. Determining how a user will interact and interface with the application is critically important to determine upfront. Create user flows and wireframes to help determine all screens that need to be designed, guide architecture, and help plan quality control tests. A user flow is represented as a flowchart detailing all the possible steps a user may go through in completing an activity in an application. A wireframe is a diagram of the content and navigational elements required by the user. A sample user flow and wireframe can be found in the annex of this document.

Technical Specification

After a functional specification has been written, a technical specification should be drafted indicating how the application will be architected, how it will interface with other application, etc.

Usability

Discoverability

When designing an application, consider how the user will discover that the application exists on disc. This may be as simple as a clearly labeled button on the main menu to activate the application or packaging that showcases the feature’s existence on the disc. Featured applications or frequently accessed applications should sit at a higher level on the disc’s menu and can benefit from being accessible via shortcuts or hot buttons.

Button Usage

As more titles are released, users will expect consistency from disc-to-disc on button usage. Disc usability can be enhanced by building upon existing button usage.

Menu (VK_MENU)

- Open in-movie menu
  - Used by: Warner, Universal, Paramount

Top Menu (VK_TOP_MENU)

- Open in-movie menu
  - Used by: Warner
- Open main menu
  - Used by: Universal

Return or Back (VK_BACK)

- Exit current screen and return to previous

Up, Down, Left, and Right (VK_UP, VK_DOWN, VK_LEFT, VK_RIGHT)

These keys are primarily used for navigation; however extending their functionality to also activate menus allowing users to navigate between screens is very useful.

A (VK_A)

- Launch in-band application (ie, U-Control)
  - Used by: Universal
- Zoom
  - Used by: Warner Bros.

B (VK_B)

- Bookmark a scene
  - Used by: Warner, Universal, Paramount

C (VK_C)

- Create a clip
  - Used by: Warner, Universal, Paramount

Graphic Design

Title Safe

While the screen resolution is 1920x1080 pixels, title safe is 1536x864 pixels from the center of the screen. This means that 192 pixels on the left and right, and 108 pixels on the top and bottom of the screen *might not* be displayed on the user's television. When creating designs, avoid putting user critical information outside the title safe region.

Focus and Selected Cueing

Choose focus and selected states that are obvious to the user without needing to navigate first. Navigation state should include more than a subtle change in color. Approximately 1 in 12 people have some version of color blindness so it is best to avoid color-cueing as the sole indicator, particularly red/green and yellow/blue. Consider lowering the opacity on non-focused elements and using 100% opacity on focused elements, creating a border or other graphical treatment around the focused element, or other ways to make the focused element stand out.

Pixel Buffer

The number of pixels on screen at any time must not exceed 4,147,200 pixels (two times the screen resolution). Pixel buffer is affected by all images (.png, .jpg, .mng, .cvi, .cdw) and fonts with display set to auto in all currently active applications (title application, playlist applications, and advanced subtitles). Images with visibility set to “hidden” are in the pixel buffer. If an image is large, changing an image’s display from “none” to “auto” may cause a delay in the image being displayed due to the need to decode the image. If pixel buffer is available, for larger images, you may want to change the display to “auto” earlier to allow the image to be decoded, and set visibility to “visible” to bring the image into view.

When designing applications, consider what elements can be reused onscreen. Repeated elements do not consume additional pixel buffer space.

For applications that run on static backgrounds, consider a looping video background instead of a graphic. The video uses no pixel buffer space and placing a background in video can save considerable pixel buffer space.

More information on the pixel buffer can be found in 3.2.3 of the Player and Content Guidelines (http://www.dvdforum.org/gen-hddvdguide.htm)

Development

Tools

IDE

The right tools can help you write more spec compliant code in a more efficient manner. While script files can be created in any text editor that can generate a big-endian UTF-16 encoded file, an integrated development environment like Microsoft Visual Studio or Eclipse can significantly improve productivity by providing Intellisense for ECMAscript methods and XML schemas. Additional information about adding the HD DVD XML schemas into Visual Studios can be found at: http://blogs.msdn.com/amyd/archive/2007/05/07/intellisense-it-s-a-good-thing.aspx

Version Control

Adding version control software such as Microsoft Visual Source Safe, Microsoft Team Foundation, CVS, or Subversion to your IDE is also highly recommended as it allows developers to track versions of their applications as well as a way to more effectively collaborate on development with a team of developers. More information on version control systems, integration, and terminology can be found at http://en.wikipedia.org/wiki/Revision_control.

Verifiers

An important step in application development for HD DVD is verification. Verification should be run both during and at the end of development to help with debugging as well as ensuring the final application is compliant with the HD DVD specifications. Several verifiers are available on the market including the Microsoft HD DVD Validator included with the HD DVD Interactivity Jumpstart Kit and verification tools included with authoring software or those from CE manufacturers.

Emulators and Simulators

Before creating a disc, content can and should be tested on an emulator or simulator. The Microsoft HD DVD Interactivity Jumpstart Kit contains an HDi simulator that allows developers to test their advanced application on a PC. The Microsoft HD DVD Interactivity Jumpstart Kit is a free download.

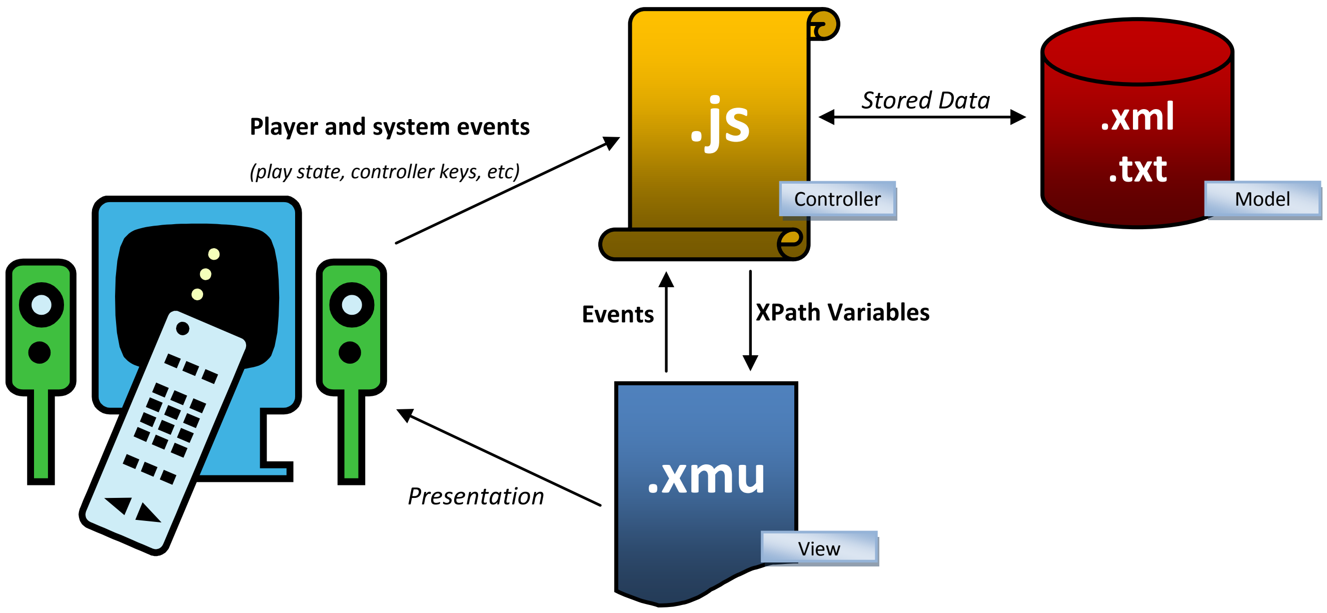
In order to test the entire project as a whole with video and content, an emulator is required. Many CE manufacturers make emulators available for their players. Information about the Xbox 360 HD DVD Emulator can be found at www.ThisIsHDDVD.com.

Architecture

Model View Controller

Complex advanced applications can benefit from the model-view-controller architecture. MVC utilizes the separation of data (model) and user interface (view) and event processor (controller).

In an HD DVD advanced application, the model could be both stored data as well as player state communicated via events. The controller is the application's script files, and the view is the application's markup.



Applications that de-couple the script, markup, and data as much as possible are more manageable, updateable, and re-skinnable. One very effective way to decouple markup and script is to avoid setting any styles via script. Whenever possible, set an XPath variable to signify to the markup that style should be updated.

Events

Applications should take advantage of the eventing model to handle user input, manage player state, and communicate with other currently active applications.

Applications should avoid overriding basic player functionality controlled via the remote such as changes to play state, trick mode, or audio or subtitle track selection. Relying on trapping keys and overriding functionality could result in applications not functioning as intended on some players because future players may introduce new features such as a new key that changes play state or a new feature that adjusts title time.   Instead, it is best to listen for player events and respond accordingly.

For example, let's suppose a developer were creating a title timeline progress bar that appears on screen whenever the video playback pauses. One approach would be to listen for the VK_PAUSE and VK_PLAY keys in script and trap default player behavior. When a user pressed the pause button, the script could call Player.playlist.pause and activate the progress bar application. When a user pressed the play button, the script could call Player.playlist.play and deactivate the progress bar. However, what happens if the user decides to fast forward? That key is not being listened for, so the application will not be activated or deactivated. Or, what if a future CE manufacturer adds a special play/pause toggle button that resumes playback without sending the VK_PLAY virtual key? We could end up in a state where the progress bar remains on screen.

A better approach would be listening for the play_state event sent by the player. When the play_state changes from play to any other state such as pause, fast forward, rewind, etc, activate the application. When the play_state changes to play, deactivate the application.

Responding to the player's events helps to ensure that the application will continue to perform as expected as more players are released to the market.

Note that the order in which events are dispatched and received may vary from player to player, so avoid order-dependent script when listening for events.

Playlist

Unlike markup, the playlist file is not extensible. Though, if information in the playlist is set correctly and in detail, the script is able to obtain very useful information. When editing a playlist, be sure to associate the HD DVD schema for playlists with your XML editor (http://www.dvdforum.org/2005/HDDVDVideo/Playlist/Playlist.xsd) to help you create a valid playlist and take advantage of all the allowed attributes and fields.

Track Navigation

While the audio and subtitle streams included in an EVOB are specified in the PrimaryAudioVideoClip tag, the user is only able to navigate to the streams mapped to a track defined in the TrackNavigationList. Likewise, the script is only aware of the streams if they exist in the TrackNavigationList.

```
<TrackNavigationList>
```

```
      <AudioTrack track="1" description="English MLP" langcode="en:01"/>
      <SubtitleTrack track="1" description="English for Hearing Impaired"
      langcode="en:01"/>
    </TrackNavigationList>
```

Use the langcode attribute to specify the language and track usage. Language is determined by the ISO 2-character abbreviation (see http://www.loc.gov/standards/iso639-2/php/code_list.php). The language code extension usage is defined in Annex B.2 of the DVD Specifications for High Definition VIDEO. Some sample usage

Audio:

- 01 - Normal
  `<AudioTrack track="1" description="English MLP" langcode="en:01"/>`
- 02 - Audio for visually impaired
  `<AudioTrack track="1" description="German for the Visually Impaired" langcode="de:02"/>`
- 03 - Directors Commentary
  `<AudioTrack track="1" description="Director's Commentary in Spanish" langcode="es:03"/>`

Subtitle:

- 01 - Normal
  `<SubtitleTrack track="1" description="English" langcode="en:01"/>`
- 02 - Enlarged captioning
  `<SubtitleTrack track="1" description="Enlarged French" langcode="fr:02"/>`
- 0D - Directors Commentary caption
  `<SubtitleTrack track="1" description="Director's Commenary" langcode="en:0D"/>`

Scheduled Control List

Scheduled Event

A scheduled event can be used to alert the script that a time in the playlist has been reached.

```
<ScheduledControlList>
    <Event id="loop_title11" titleTime="00:49:59:00"/>
</ScheduledControlList>
```

The id for a scheduled event, as with all ids in the document, must be unique per playlist. Do not use a scheduled event during the first or last 10 frames of a title as it may be missed.

Scheduled events will only fire at a single moment in time, are not guaranteed to fire during trick play or chapter jumping. For events that should be fired during a given time frame, use markup timing with a title clock in a title application. For example, the display of picture-in-picture needs to be available during a span of time, so a markup timing element would be more appropriate.

Scheduled Pause

The scheduled control list can also contain a scheduled pause. When using an EVOB with a static background, you can use a scheduled pause rather than calling pause from the script.

```
 <ScheduledControlList>
```

```
   <PauseAt id="pause_title1" titleTime="00:00:01:00"/>
</ScheduledControlList>
```

Do not use a scheduled pause during the first or last 10 frames of a title as it may be missed.

Markup

XML Schemas

When writing markup files, take advantage of the HD DVD markup, manifest, and playlist schemas to help you create a valid playlists, markup, and manifests and to take advantage of all the allowed attributes and fields.

- http://www.dvdforum.org/2005/ihd/iHD.xsd
- http://www.dvdforum.org/2005/ihd/iHDstate.xsd
- http://www.dvdforum.org/2005/ihd/iHDstyle.xsd
- http://www.dvdforum.org/2005/HDDVDVideo/Manifest/Manifest.xsd
- http://www.dvdforum.org/2005/HDDVDVideo/Playlist/Playlist.xsd

Add the schema namespace to your root node in markup:

```
<root xml:lang="en"
xmlns="http://www.dvdforum.org/2005/ihd"
xmlns:style="http://www.dvdforum.org/2005/ihd#style"
xmlns:state="http://www.dvdforum.org/2005/ihd#state>
```

and manifest:

```
<Application xmlns="http://www.dvdforum.org/2005/HDDVDVideo/Manifest">
```

and playlist:

```
<Playlist xmlns="http://www.dvdforum.org/2005/HDDVDVideo/Playlist"
majorVersion="1" minorVersion="0">
```

References to these schemas in your XML editor will help in creating more spec compliant markup.

Additional information about adding the HD DVD XML schemas into Visual Studios can be found at: http://blogs.msdn.com/amyd/archive/2007/05/07/intellisense-it-s-a-good-thing.aspx

XPath Expressions

When accessing id or class attribute in an XPath expression, use the id(), class(), and defaultNode() methods to improve responsiveness. For example:

Given:

```
<button id="my_button" class="special"/>
```

DO NOT Use:

```
//button[@id='my_btn' and state:focused()]
```

Should Use:

```
id('my_btn')[state:focused()]
```

DO NOT Use:

```
//button[@class='special' and state:actioned()]
```

Should Use:

```
class('special')[state:actioned()]
```

Use defaultNode() to returns the first node of the nodeset returned in the begin

DO NOT Use:

```
<cue begin="class('special')[state:actioned()]" end="class('special')
[not(state:actioned())]"/>
```

Should Use:

```
<cue begin="class('special')[state:actioned()]"
end="defaultNode()[not(state:actioned())]"/>
```

When using larger number of XPath evaluations in a timing element, group them in a hierarchy such that they are not all executed at every tick. For more information, see Nest Divs and Pars under Markup Performance.

XPath Variables

Use XPath variables to communicate state changes to markup from script. An XPath variable can be set in script using setXPathVariable.

```
 document.setXPathVariable("CUSTOM_VAR","foo",);
```

In markup, the XPath variable is prefaced with a dollar sign ($) and the value is always a string. The evaluation should utilize parentheses.

```
 <par begin="($CUSTOM_VAR = 'foo')" end="($CUSTOM_VAR != 'foo')">
   <!-- put cues here -->
 </par>
```

When using XPath variables, be sure to initialize the value in the markup loaded handler.

Timing

Markup styles and states can be modified entirely by cues in the markup's timing element. If timing is tied to the movie timeline, a title clock should be used where the begin, end, and dur attributes are time values. Title clocks may only be used in title applications, never in a playlist application.

Playlist applications or title application that use XPath expressions and variables must use a title or application clock.

For more information on using the timing element, clocks, and cues, read:
http://blogs.msdn.com/amyd/archive/2007/06/26/making-the-most-of-markup.aspx

Extending Markup

The markup is extensible and custom attributes can be added if you add the namespace to the root element of the document.

```
xmlns:custom="uri://custom"
```

and adding an attribute from this namespace to a markup element

```
custom:onleft="doSomething"
```

This allows us to pass custom information on the button that has been the target of an action to the script by calling getNamedItem on the target of the action.

```
var element = evt.target;
var attr = element.attributes.getNamedItem("custom:onleft");
var key = attr.value;
```

Additional information on the XML API can be found in the DOM Level 2 Core Specfication:
http://www.w3.org/TR/DOM-Level-2-Core/core.html

Element Positioning

Use absolute positioning where possible as static or relative can have unpredictable results. The elements <div>, <button>, and <object> can be positioned, meaning you can specify a value for the x and y coordinates onscreen. The elements <input> and <p> not positionable and must be placed inside a <div> for positioning.

When absolutely positioning an element, always specify width. The elements <button>, <object>, and <input> always require both height and width regardless of positioning.

Use pixel values rather than percentages for clarity. Some size and positioning will not work if a percentage value is used. Screen resolution will always be a fixed height and width based on region value specified in the manifest.

Pixel Buffer

The number of pixels on screen at any time must not exceed 4,147,200 pixels (two times the screen resolution). For more information on the pixel buffer, read the Pixel Buffer section found under Graphic Design.

Markup Performance

Nest Divs and Pars

Nest divs and pars in markup to improve performance. When using larger number of XPath evaluations in markup, group them in a hierarchy such that they are not all executed at every tick. For instance, if a parent menu contains child buttons, the pars which handle focus and action of those buttons should only begin once the parent menu is active and should end once the parent menu is no longer visible.

Instead of:

```
 <par>
    <cue select="id('my_button')" begin="id('my_button')[state:focused()]"
end="defaultNode()[not(state:focused())]"/>
</par>
```

Use:

```
<par begin="id('parent_menu')[style:visibility()='visible']"
end="defaultNode()[style:visibility()='hidden']">
    <cue select="id('my_button')" begin="id('my_button')[state:focused()]"
end="defaultNode()[not(state:focused())]"/>
</par>
```

IDs

Only use the id attribute on elements that your timing element or script will need to reference. Avoid giving every element an id.

Avoid referencing non-existent ids in the timing element. Be sure that every element reference in the timing block is valid.

General Markup Tips

Separation of UI and logic

When possible, avoid modifying the style of markup elements via script. By de-coupling UI (markup) and logic (script), applications are easier to debug, re-skin, and extend. For more information, see the model-view-controller section.

Implicit Navigation

Take advantage of implicit navigation when possible. The player will compute implicit navigation of navigable elements on screen based on their x, y positioning. Navigation can specified explicitly by using navUp, navDown, navLeft, navLeftUp, navLeftDown, navRight, navRightUp, navRightDown, navIndex in the style namespace as defined in 7.6.3.3.2.32 through 7.6.3.3.2.40 of the spec.

If the on-screen markup changes, computerImplicitNav() will need to be called in the script in order to continue using implicit navigation with the updated user interface. See http://blogs.msdn.com/amyd/archive/2007/10/24/virtual-keyboards.aspx for an example of implicit navigation and how computerImplicitNav() is used.

accessKey

Instead of listening for controller_key_down for special keys in the script, use the accessKey attribute on markup elements.

Minimize the total number of elements and XPath expressions whenever possible and consider using multiple markup files and linking between them.

Use <include> for common sections to reduce maintenance costs

Input

The input tag has four options for mode: password, singleline, multiline or display. When an input element with singleline, multiline, or password mode receives focus, the player ***may*** launch an onscreen keyboard to allow user input or a software player may enable use of the computer keyboard.

On <input> elements meant for display only make sure to set the mode attribute to display or the style:navIndex attribute to "none" to prevent the element from receiving focus during implicit navigation.

If you have <input> elements, you must define a font even if the elements have no values.

Additional Reading

For more information on using markup, read:
http://blogs.msdn.com/amyd/archive/2007/06/26/making-the-most-of-markup.aspx

Script

Exception Handling

The player may throw exceptions during the execution of script, and if not caught will result in a player crash. Specific information about the conditions under which an exception may be thrown is detailed for each API defined in Annex Z of the DVD Specifications for High Definition Video.

Before making a call to an API, check conditions under which an exception may be thrown. For example, calling XMLParser.parseString will throw an exception if XMLParser.status is not equal to XMLParser.READY, so check status before proceeding.

Released titles must use try-catch exception handling in all entry points including all global code, event handlers, and function callbacks. While specific exceptions will be thrown based on conditions, do not write script that relies on specific exceptions. Keep try blocks to small chunks of code and attempt recovery wherever possible.

```
try
{
    if (XMLParser.status() == XMLParser.READY)
    {
        XMLParser.parse("file:///dvddisc/ADV_OBJ/data.xml",
        parseCallback);
    }
    else
    {
        //parseFailed would be a function that proceeds appropriately
        //when success path can not be followed
        parseFailed();
    }
}
catch (ex)
{
    //utility from "Helpful Tracing Routines" that writes diagnostic info
    TraceError("EXCEPTION", ex, arguments.callee);
    //parseFailed would be a function that proceeds appropriately
    //when success path can not be followed
    parseFailed();
}
```

Because using try-catch may make debugging difficult, consider using conditional compilation around calls to try-catch. Conditional compilation is a proprietary Microsoft extension for Jscript that can be used during development with the Microsoft HDi simulator from the Interactivity Jumpstart Kit, the Xbox HD DVD emulator, and some software players. For more information on using conditional compilation during HDi development, read: http://blogs.msdn.com/ptorr/archive/2007/06/01/using-conditional-compilation-in-hdi.aspx

As demonstrated above, all catch blocks should also contain diagnostic trace information to further assist during development. For more information on helpful tracing routines, read: http://blogs.msdn.com/ptorr/archive/2007/06/27/helpful-tracing-routines.aspx

Timers

Default values

By default, timers are repeating timers (autoReset = "true") and disabled (enabled = false). When creating a one-shot timer, autoReset must be set to false. The timer's countdown will not begin until enabled is set to true.

Closures and Circular References

Misuse of timers and closure functions are often a source for player memory leaks. For more information on these terms, read the section on Closures and Circular References in Script Performance. To avoid memory leaks resulting from timers:

- Avoiding closures altogether
- Explicitly break the cycle by setting the referring timer to null
- Avoid making the timer a local variable

Re-use of Timer Variables

When createTimer is called, a timer object is created and a reference to that object is returned. If a new timer is created and the same script variable is used, the reference to the first timer is broken, but the timer object still exists and may fire. The simple solution here is to always use a unique name for your timers, or ensure they are local variables that won't conflict with other timers. If re-using a variable name that previously held a timer reference, disable the first timer by setting enabled to false before re-using the variable.

General Misuse of Timers

Timers are often used when a more elegant, less memory intensive solution is available in HDi.

Incorrect use of a timer: Performing many set operations on a repeating 1-frame timer

Preferred HDi alternative: Use the <animate> element in markup or the animateProperty API in script

Incorrect use of a timer: Constantly polling currentTitle.elapsedTime to check for a specific timecode

Preferred HDi alternative: Use an <event> in markup or a PauseAt or Event in the playlist's scheduled control list.

For more information on using timers in an HD DVD application, read:
http://blogs.msdn.com/ptorr/archive/2007/08/29/using-timers-effectively-in-hdi.aspx

Set/Unset Property

Script has the highest priority for markup properties and will override markup and user input. By modifying the markup styles and state by using setProperty() and animateProperty() script takes control of the property until unsetProperty() is called. For example, if setProperty is called on focus in the script, the user will not be able to navigate until unsetProperty is called.

For more information on using setProperty and unsetProperty, read:
http://blogs.msdn.com/ptorr/archive/2006/09/08/746678.aspx

For more information on using markup to control style, read:
http://blogs.msdn.com/amyd/archive/2007/06/26/making-the-most-of-markup.aspx

Debugging

Use the Diagnostics object to trace program flow.

Use onscreen tracing for immediate feedback by setting the value of an input element.

Use the Microsoft HDi simulator with -jit in the command line and the Microsoft Script Debugger to step through script code.

Use the Xbox HD DVD Emulator PC logger to monitor script and markup execution.

For more information on helpful tracing routines, read:
http://blogs.msdn.com/ptorr/archive/2007/06/27/helpful-tracing-routines.aspx

Persistent Storage

Because the players' persistent storage is flash memory, avoid frequent or unnecessary access as flash based memory has a finite number of erase-write cycles.

Provide users with the ability to modify the contents saved on persistent storage for your disc from outside the player's persistent storage management menu. For instance, if the disc allows for the downloading trailers, make sure it also provides users with the ability to delete these trailers from persistent storage via an application on the disc.

Label content and provider folders by setting a *[language]-explanation* as specified in chapter 10 of the DVD Specification for High Definition Video. When a user does use the player's persistent storage management menu, the disc's folder will be appropriately labeled and the user can avoid confusing it with another disc and accidentally deleting it. The *[language]-icon* key may also be set in addition, but should not be the only label for a persistent storage folder.

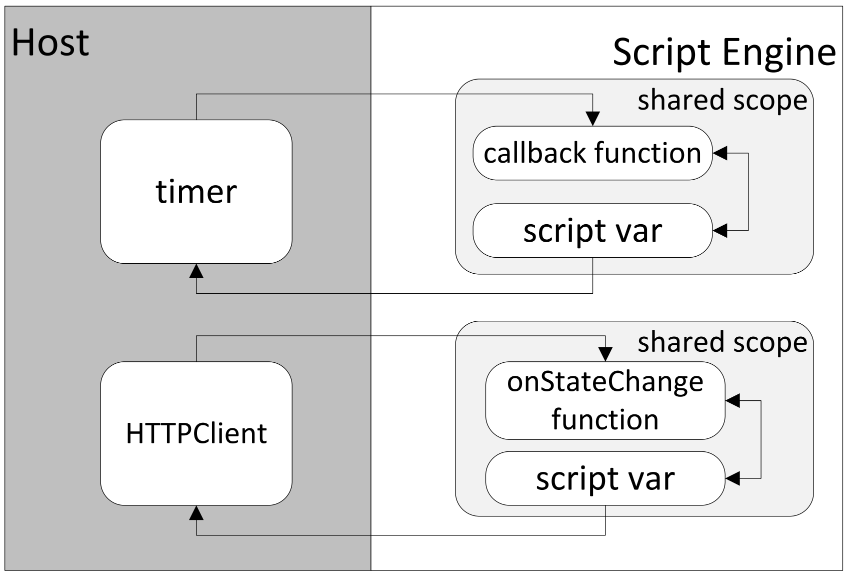
For more information on labeling persistent storage, read:
http://blogs.msdn.com/amyd/archive/2007/08/15/persistent-storage-management.aspx

Script Performance

Closures, Circular References, and Memory Leaks

A "closure" is a nested function in ECMAScript that maintains a reference to all the local variables of its containing function. This reference "closes it off" from being released for garbage collection. Normally this isn't a problem for ECMAScript, because sooner or later the closure dies and the other objects it has been keeping references to will also die.   However, if the closure includes a reference to a Host Object, such as a timer or an HTTPClient object a circular reference is made.

When a circular reference is made with a Host Object, garbage collection is unable to reclaim the memory even after the function has completed execution because the script engine has no knowledge of the state of the host object it is referencing. As a result, memory is leaked. A significant number of these memory leaks will result in a player crash.



In order for the script object to be garbage collected and memory freed, the reference to the host object must be explicitly broken when scope is shared.

When using a timer or HTTPClient, always review the script for possible memory leaks. Avoid testing by simply running the application and waiting for the player to crash.

For more information on ECMAScript closures and circular references, read:
http://blogs.msdn.com/ptorr/archive/2007/08/29/using-timers-effectively-in-hdi.aspx and
http://msdn2.microsoft.com/en-us/library/bb250448.aspx

Other Memory Usage Issues

In general, host objects such as XML DOMs and HTTP responses should be used sparingly.

When communicating with the server, if a response is not required, communication done via headers uses less memory. Avoid sending a response from the server if one is not needed.

If a response is sent from the server and not downloaded, call getResponseString on the HTTPClient to free memory used by the response.

Cache markup references

When frequently modifying the state or style of a markup element via script, cache a reference to the element's script or style.

ex:

```
var inputState = document.getElementById("my_input").style;
inputStyle.color = "red";
```

Pre-compile and Cache Data

When data must be stored in XML, cache values in script whenever possible to reduce the frequency that the DOM must be accessed. Avoid per-tick work if you can front-load it. Rather than reading in data at runtime, consider generating pre-populated script variables as part of your build process.

Asynchronous Code

APIs that use callbacks introduce a complexity that is different than web development and benefits from the "event driven" approach. Beware of race conditions between events and callbacks as the timing upon which the callback will occur cannot be guaranteed.

When handling the callback status or error code, the only reliable callback value is for success. Other status or error values may give you insight into what has occurred, but should not be relied upon for critical paths.

General Script Tips

Separation of UI and logic

When possible, avoid modifying the style of markup elements via script. By de-coupling UI (markup) and logic (script), applications are easier to debug, re-skin, and extend. For more information, see the model-view-controller section.

Simplify Design

Wherever possible, use simple designs and write short, simple, functions. Avoid tight coupling between functions. Use descriptive identifiers.

Miscellaneous

Comment code appropriately to assist both others and yourself.

Leave an empty line at the end of each script file to prevent errors when player concatenates script files.

Script files must be UTF-16 Big Endian.

There is no automatic semi-colon insertion.

Data and XML

Storing data in xml is a convenient way to organize it for easy retrieval. However, parsing a DOM, retrieving elements, getting and setting attributes can prove to be memory intensive.

When data needs to be saved to persistent storage, consider reading and writing keys to info.txt using getContentInformation and setContentInformation where appropriate.

For data being pulled from the disc, consider using a preprocessor to generate script files with variables preset.

When data must be stored in XML, cache values in script whenever possible to reduce the frequency that the DOM must be accessed.

Networked Content

Downloading Files

If downloading content that will be saved to persistent storage, make sure it also provides users with the ability to delete these trailers from persistent storage via an application on the disc. Also, be sure to set the [language]-explanation tag so that your content is clearly identifiable by the user when accessing the persistent storage management menu.

When downloading a file to persistent storage or filecache, always check the size of available space first and do not proceed with the download if space is not available. Because a download may fail or be aborted, downloaded files should be saved with a temporary name and renamed to the final filename once all related downloads are successful.

Using Alternative Playlists

It is best to ship a disc with a variety of playlists or a playlist with a variety of titles that can accommodate different EVOB lengths and application sizes. A playlist may have up to 999 titles which allows for a lot of flexibility with EVOBs of different duration, subtitle and audio information, and associated applications and advanced subtitles.

Booting an alternative playlist from disc rather than persistent storage does not require additional AACS keys to be downloaded, which saves on persistent storage space, download times, and key generation fees. In addition, automatically booting a playlist from persistent storage may prove to be problematic if resources required by that playlist are not available due to a failed download or intentional deletion by the user.

If downloading new content that does require a new playlist to be downloaded as well, make the playlist boot/re-boot more resilient. If a playlist fails to boot properly, recovery is not possible like a script failure might be by using exception handling. So, before booting a new playlist, write a key to info.txt specify the file name of the playlist that is attempting to boot. On successful re-boot, the key can be deleted. On an unsuccessful re-boot, the user will be force to manually restart the player. On the subsequent re-boot, a default playlist should load which reads the key to determine the unsuccessful playlist boot and handles the failure appropriately by falling back to a “last known good configuration”. The sample user flow in the annex of this document details this process.

General Usability Issues

When downloading files, provide the user with a progress indicator displaying percentage complete and a button to allow cancellation of the download. If a user cancels a download or the connection fails, be sure to delete associated files that are incompatible with the failed or aborted download.

Avoid using the HTTP user-agent string to determine the model of player and serve specialized content as this value may change.

Preferably, connections should be made on the request of the user rather than automatically. Automatic downloads at the disc boot may be problematic and prevent the user from being able to watch the movie.

While the player may have up to 8 simultaneous network connection, serial downloads are preferable to concurrent as error handling and user response are more easily managed when dealing with one connection.

When downloading files, use an indefinite timeout (timeout value of -1). The throughput value of a player may not be accurate or may change over the course of a download and thus calculating a timeout based on this value may result in downloads timing out before download completes. To ensure that download progress has continued when using an infinite timeout, the dataDownloaded value can be monitored periodically.

Memory Issues

The circular reference and memory leak issue common with timers, also exists with the HTTPClient object. Explicitly break the cycle by setting the referring HTTPClient to null when the connection is complete if it is not being reopened. For more information on this topic, read the section on Closures and Circular References in Script Performance.

When communicating with the server, if a response is not required, communication done via headers uses less memory. Avoid sending a response from the server if one is not needed.

If a response is sent from the server and not downloaded, call getResponseString on the HTTPClient to free memory used by the response.

Testing

Testing Strategy

Planning the test strategy prior to actual testing will often discover issues with the specification and eliminate defects before they are coded. Writing a test plan will help clarify the goals of testing, as well as calling out what will and will not be tested. Test plans should be specific to each title, since there will be variations in the functionality for each title. Having a test plan template that covers many of the common features is a good starting point. The template can then be customized to address title specific features.

The test plan should touch upon all features in the title. Consider all possible user scenarios that may occur. Consider areas for vulnerability for failure and test how the application responds. The test plan should include the strategy for testing the following areas:

- Audio / Visual quality
- Menu functionality
- Audio and subtitle tracks
- Special feature content (Deleted scenes, trailers, games, Picture in Picture, etc)
- Persistent storage testing

- Network and server testing
- Players that will be tested (HD-XA1, HD-A2, Xbox 360 add-on player, etc)

Once the test plan has been written, it is recommended that representatives from test, design, and development meet to review the test plan. This will give all appropriate stake holders the chance to review the test strategy and hopefully point out any holes in the test plan, redundancies, or recommendations for further testing.

Test Cases

It is beneficial to begin building a set of Build Verification Test (BVT) cases during development. As features are completed, test cases should be written that validate that the feature functions per spec. These test cases don't need to fully test the feature (those test cases should be implemented later), but they should validate that the feature functions correctly. As builds are produced, running the BVT cases will ensure that subsequent check-ins do not break the functionality that has already been implemented.

It is recommended that tests are run on a variety of platforms during development. Building code in small chunks and running unit tests on those pieces as each is complete will help in identifying problems as well as creating a more smooth integration process with the other features in the title.

Functional test cases should also be written during the development phase. The functional test cases differ from the BVT cases in that they should validate the robustness of the title. The functional test cases should be thought of as a superset of the BVT cases. While the BVT cases validate the baseline functionality of the feature, the functional test cases should validate completeness of the feature (all code paths are exercised), and should also test negative scenarios. Negative test cases are needed to ensure that the title will successfully handle scenarios where the system receives input it is not expecting. An example of this would be if a user attempts to access online content, but can't successfully reach the server. The correct behavior would be to display an appropriate message and allow the user to return to the menu, as opposed to displaying an error code requiring a reboot of the player.

Test Areas

Once all the required features are coded and implemented, this is when a full test pass including all end to end scenarios should be executed. This includes server content and all network scenarios. Below is a list of areas that should be validated:

- **Player testing**. In order to ensure compatibility with all players on the market, the test setup should include one of each player with the latest firmware. Before the disc is certified as ready for replication, a test pass should be run on each player.
- **Menu Testing.** All menu items should be verified. The menu items should show correct focus, and produce the desired result when exercised. All points of entry to the menu should be verified (i.e. accessing menu items from title menu as well as the in-movie menu). The user should be able to access all menu items (scenes, setup options, special features) in one session without rebooting the player. Text and font rendering will vary from player to player. Sizing/clipping of any dynamic should be closely reviewed.
- **Audio/Video Testing**. Given that there are often several subtitle and language tracks included on the disc, each audio and subtitle track should be verified. It is recommended that the entire movie be played with subtitle and language tracks enabled to ensure that the audio and video stay in synch throughout the movie.

- **Special Features testing**. All trailers, deleted scenes, and other special features should play successfully and return to the menu upon completion. All content in “Deleted Scenes”, “Trailers”, etc should be played from start to finish. In the case of a “Play All” option, each clip should play successfully, upon completion the next clip should play, and after the last clip completes, the menu should be displayed. Items like Picture-In-Picture should function correctly when doing fast forward, fast reverse, and chapter skip operations.
- **Persistent Storage testing**. Features such as creating bookmarks, clips, downloading content from the network, or saving data to the hard drive may require writing to the persistent storage. If possible, the data written to the persistent storage should be labeled so that the user can easily identify and delete the content. The player should give the correct notification when the content to be added (bookmark, clip, downloadable content) will not fit in the space available.
- **Network testing**. Network functionality is one of the key differentiators between HD DVD and Blu-Ray, so it is imperative that this functionality is well tested and provides the user with a robust experience. All content available on the servers should be downloaded and viewed. In addition, the player should be rebooted and the downloaded clips should be viewed again to ensure the integrity of the content in persistent storage. It is recommended that size of each download item and the time required to complete the download be recorded. Having this data will help troubleshoot issues in the event that users experience problems with this feature.

  It is very important to test unsuccessful download scenarios. These would entail the user cancelling the download as well as network related issues (low network bandwidth, connection disconnected, etc). If the download is unsuccessful, the next attempt to download the content should result in the player detecting that there are residual files from the previous attempt and handle the situation accordingly.
- **Server testing.** It is recommended that the server be set up and all players can access the server correctly. It is advised that a staging server be available to internally test all content prior to making it available to the general public. The server content should be updated, and the players should be able to see and download the new content successfully.

Resources

Forum

The HD DVD authoring forum, hosted on MSDN, is an excellent way to get answers to your technical questions about authoring HD DVD interactive content.

http://forums.microsoft.com/MSDN/ShowForum.aspx?ForumID=527&SiteID=1

Blogs

Read about HD DVD topics such as script writing, debugging, and how to add advanced features, as well as practical tips and tricks from the HD DVD team at Microsoft.

Peter Torr: http://blogs.msdn.com/ptorr

Amy Dullard: http://blogs.msdn.com/amyd

Andy Pennell: http://blogs.msdn.com/andypennell

White Papers

Introduction to HD DVD Authoring: http://msdn2.microsoft.com/en-us/library/aa468611.aspx

Planning and Producing HD DVDs: http://thisishddvd.com/Downloads/MakingHDDVDs3.pdf

Tools and Samples

HD DVD Interactivity Jumpstart Kit:
http://www.microsoft.com/downloads/details.aspx?FamilyId=F8ADA3F5-0EC6-4392-84AB-CB4860DB30ED&displaylang=en

HD DVD Interactivity Sample Code:

http://www.microsoft.com/downloads/details.aspx?FamilyId=E19C869B-EF37-4E5E-91ED-32D059E3775B&displaylang=en

Microsoft Visual Studio:

http://msdn2.microsoft.com/en-us/vstudio/default.aspx

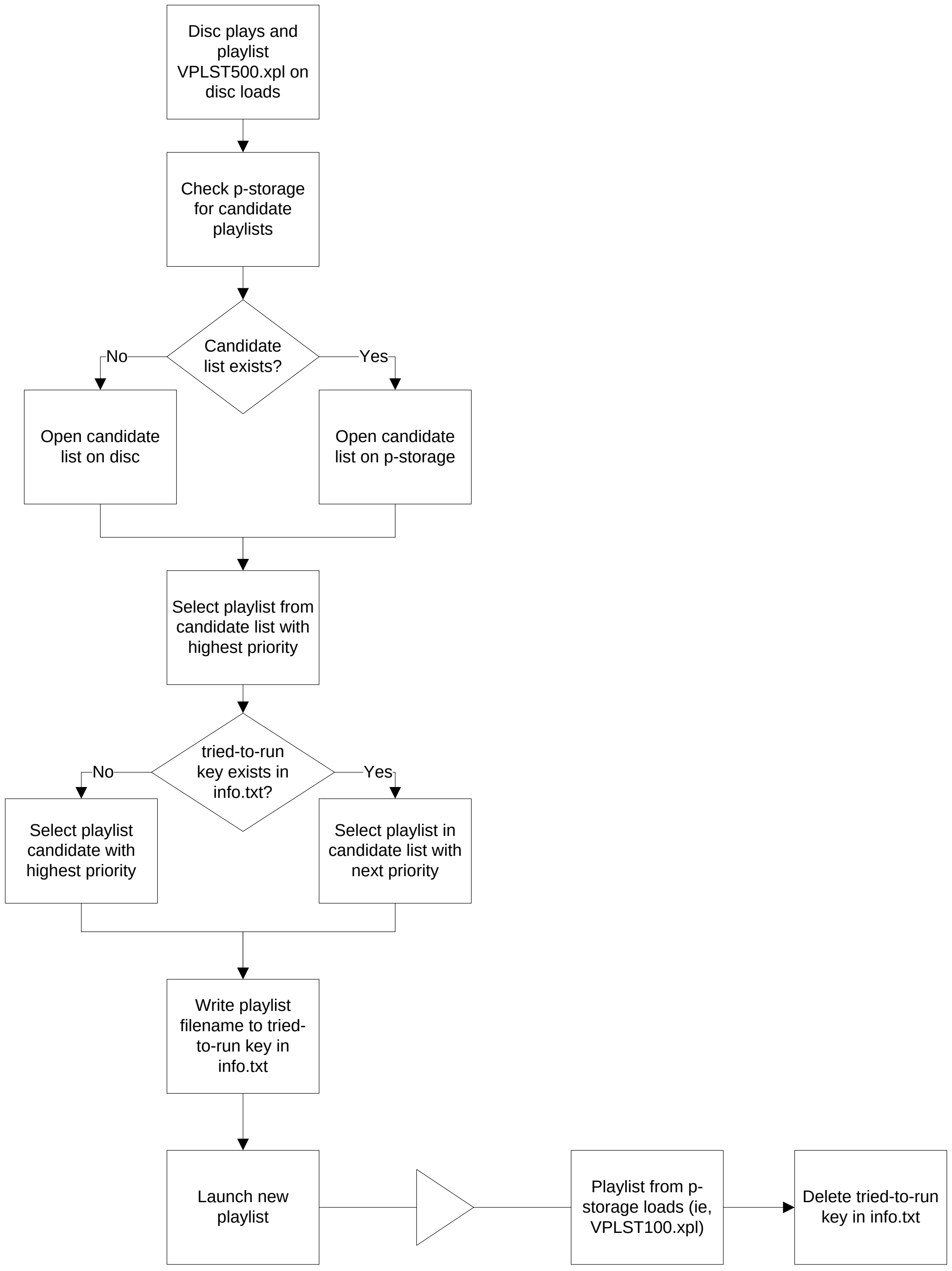
Microsoft Visual Web Developer – Express Edition (free):

http://www.microsoft.com/express/vwd/

Annex

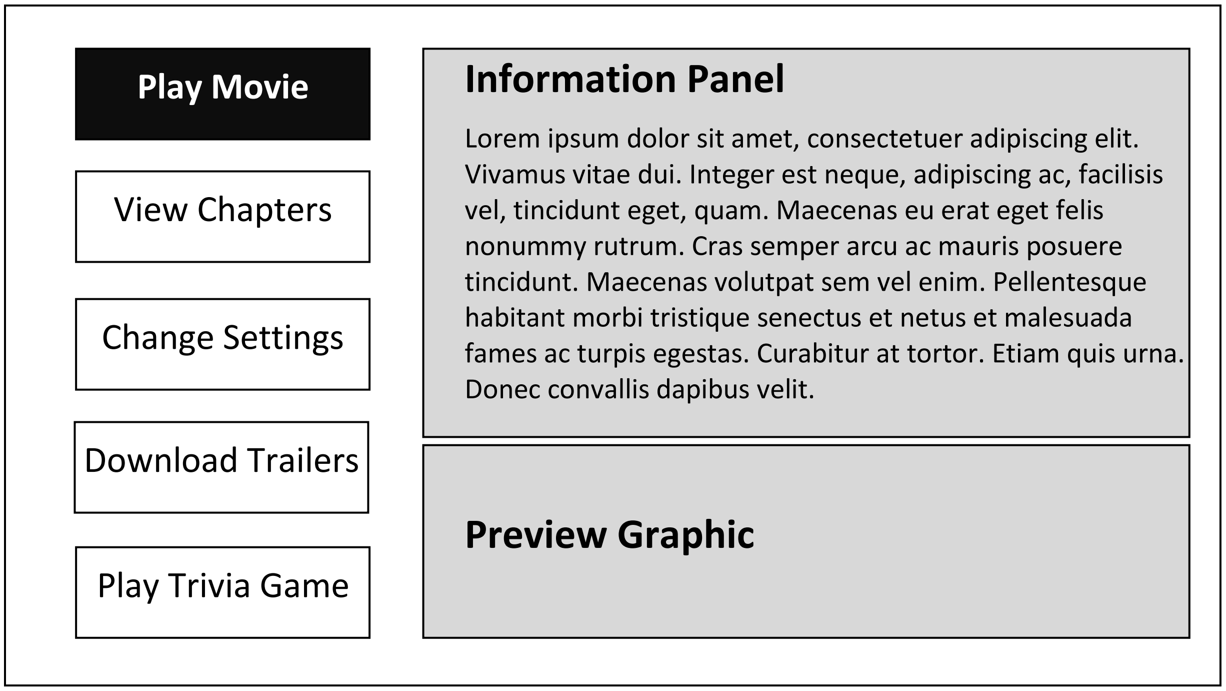
Sample User Flow

A user flow is represented as a flowchart detailing all the possible steps a user may go through in completing an activity in an application. Below is a sample workflow for candidate playlist booting.


Disc plays and playlist VPLST500.xpl on disc loads
Check p-storage for candidate playlists
Candidate list exists?
No
Yes
Open candidate list on disc
Open candidate list on p-storage
Select playlist from candidate list with highest priority
tried-to-run key exists in info.txt?
No
Yes
Select playlist candidate with highest priority
Select playlist in candidate list with next priority
Write playlist filename to tried-to-run key in info.txt
Launch new playlist
Playlist from p-storage loads (ie, VPLST100.xpl)
Delete tried-to-run key in info.txt

Sample Wireframe

A wireframe is a diagram of the content and navigational elements required by the user. A wireframe should not address visual design.


Play Movie
View Chapters
Change Settings
Download Trailers
Play Trivia Game
Information Panel
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Vivamus vitae dui. Integer est neque, adipiscing ac, facilisis vel, tincidunt eget, quam. Maecenas eu erat eget felis nonummy rutrum. Cras semper arcu ac mauris posuere tincidunt. Maecenas volutpat sem vel enim. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Curabitur at tortor. Etiam quis urna. Donec convallis dapibus velit.
Preview Graphic