

# **Mulberry Slideshow XML (v 2.1): A User's Guide**

Mulberry Technologies, Inc.  
17 West Jefferson St., Suite 207  
Rockville, Maryland 20850  
voice: 301/315-9631  
fax: 301/315-8285  
[info@mulberrytech.com](mailto:info@mulberrytech.com)

May 22, 2001



**Mulberry  
Technologies, Inc.**

<b>CREATING AND EDITING SLIDE SHOWS WITH THE MULBERRY SLIDESHOW PACKAGE .....</b>	<b>4</b>
<b>Summary .....</b>	<b>4</b>
Package.....	4
Features.....	5
<b>Slideshow system files and setup .....</b>	<b>5</b>
DTD and associated files .....	5
Your DOCTYPE declaration – the key to slideshow portability .....	5
Associating your XML file with a stylesheet.....	5
<b>The slideshow document model .....</b>	<b>6</b>
Dates, versions, and the change log .....	6
The structure of a slideshow .....	6
Notes in slideshows .....	11
Exhibits, exercises and appendixes .....	11
Cross-referencing.....	12
Regular, overview and “deep” slides .....	12
<emph> and its attributes.....	13
Specialized inline elements.....	14
<b>Graphics in Slideshows.....</b>	<b>15</b>
<b>Going Into Production.....</b>	<b>15</b>
Fixing up the graphics.....	15
The Jade engine .....	15
Using XSLT instead.....	16
Batch file processing.....	16
Manual finishing.....	16
Customizing colors .....	17

Copyright  
=====

Copyright (c) 2001 Mulberry Technologies, Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

Except as contained in this notice, the names of individuals and companies credited with contribution to this Software shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization from the individuals in question.

Any tools, document models, stylesheets, or other components derived from this Software that is publicly distributed will be identified with a different name and the version strings in any derived Software will be changed so that no possibility of confusion between the derived package and this Software will exist.

Warranty  
=====

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL MULBERRY TECHNOLOGIES, INC., OR ANY OTHER CONTRIBUTOR BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

## Creating and Editing Slide Shows with the Mulberry Slideshow Package

The Mulberry Slideshow document type was developed to support the creation of courses, tutorials and presentations that Mulberry Technologies delivers to clients and at conferences. Several shortcomings of industry-standard (but proprietary) formats for presentations originally motivated its development. Since many of our courses cover similar material, and yet are custom-designed to the needs of a particular client or audience, we wanted a format that would facilitate long-term maintenance and reuse of materials. We wanted to be able, when necessary, to publish across media: to create, for example, reasonably nicely formatted print as well as on-screen display. We wanted to be able to put courses or presentations onto the web. We wanted to be able to create and control materials that would accompany a course, but that would not be part of the presentation itself, such as printed exhibits or a set of handouts with instructions for hands-on exercises. Finally, we wanted a format that would provide us with a measure of high-level control of the material by organizing it into modules or segments – and yet we wanted it to be possible to hide that organization. That is, although a course might be structured internally, we wanted it to be up to the course developer to decide how much of that “skeleton” would be exposed to the audience, since in our experience, such exposure can sometimes be a help, sometimes a distraction.

What we have ended up with, we think, is an interestingly powerful and versatile (if unconventional) document type, which scales nicely to support everything from brief presentations of ten or twelve slides, to whole courses taking three or four days to deliver. Partly since we are designers of documentary information systems, not of layouts or presentations, the resulting presentations have never looked especially glamorous. (We console ourselves with the notion that it *could* look good if a “real designer” could provide us with specifications for our stylesheets.) But they have worked, we have found, quite well. The system has certainly succeeded in providing us the advantages of XML: data reuse, multiple output formats, freedom in tools selection, and functionalities over and above what is typically delivered in off-the-shelf software packages.

To use this package, you will need a basic understanding of XML systems and processing, and some skill with XML tools. It is not intended as a “learning application” for users just getting started with the technology.

On the other hand, as a demonstration, this package *is* intended as a jumping-off point. Please feel free to alter this application as you see fit, to do what you need it to do. We are also interested in any feedback you feel able to give us: send e-mail to **Error! Bookmark not defined.**

### Summary

#### Package

This package includes:

- XML DTD with modules
- DSSSL stylesheets to produce print and web output
- XSLT stylesheets to produce web output, both authoring and final
- CSS stylesheet for tinkering with colors in web output
- Configuration and stylesheets for the SoftQuad XMetaL editor
- Various supporting files (examples, graphics etc.)

See the file `readme.txt` for details.

## Features

A slideshow is composed of slides, each of which makes a separate page for display. Slides may be grouped in segments; the model is recursive, so segments may be nested to any depth. A segment may have any of three different types of heads, each of which has a distinct behavior in the output processing provided. This feature enables the user to create a slideshow whose intellectual organization is exposed to the audience, or one whose organization is only implicit (yet still useful for creation and maintenance).

Each slide has a title and a body. Notes may be included, but do not appear in output. The body of a slide may contain lists, tables, and graphics images. Elements that may appear inside paragraphs or list items include blocks of code. Inline elements include code, emphasis, and several elements supporting indexing, linking and cross-referencing. Specialized elements may also be used to create print handouts to accompany a presentation.

Production routines create print output (DSSSL stylesheet with RTF output) and web output (either a DSSSL or an XSLT stylesheet may be used to create a linked sequence of web pages), each with a table of contents. Additionally, an XSLT stylesheet can be used to create an outline view of the entire presentation.

## *Slideshow system files and setup*

### DTD and associated files

The Slideshow DTD and its associated entity files and modules are:

```
slideshow.xml.dtd
ISolat1.pen
ISolat2.pen
ISONum.pen
ISOpub.pen
ISOtech.pen
xmldata.ent
```

Version 2.1 is the current version as of May, 2001 (slated for an upgrade).

### Your DOCTYPE declaration – the key to slideshow portability

Assuming the DTD and all its modules can be found in a DTD subdirectory directly below the document, you should be able to validate slideshow documents with the XML processor of your choice as long as their DOCTYPE declarations include the public identifier

```
PUBLIC "-//Mulberry//DTD Mulberry Technologies Slide Show XML DTD
v2.1 19990901//EN" "DTD/slideshow.xml.dtd"
```

or the system identifier

```
SYSTEM "DTD/slideshow.xml.dtd"
```

You may also, of course, copy the DTD and associated files to your own local subdirectory for standalone processing: in this case, tweak your DOCTYPE declaration accordingly.

If you use a system that resolves the DTD through an OASIS catalog lookup, the file `xmldsssl.soc` in the DSSSL subdirectory may be used to dereference the public identifier (it may have to be edited for your system).

### Associating your XML file with a stylesheet

If you are using SoftQuad XmetaL, creating or opening a document associated with the `slideshow.xml.rlx` rules file will automatically bring in the XMetaL stylesheets (as long as you keep them together in the distribution).

If you are not (or even if you are), we have also provided an XSLT stylesheet that creates HTML output, designed to provide an author's view of the document. Rather than creating a linked web of pages, output appears with all slides appearing in sequence on a single page.

You can either use this stylesheet with an XML/XSLT browser, such as Microsoft Internet Explorer installed with XSLT support (MSXML3 or later: for information about this see the MSDN web site and/or the MSXML FAQ at [www.netcrucible.com](http://www.netcrucible.com)), or use it with a standalone processor.

The sample document included with this distribution, `tagzoo.slides.xml`, is configured to use this stylesheet. The processing instruction:

```
<?xml-stylesheet href="XSL/slides.author.xsl" type="text/xsl"?>
```

indicates this association (the href link must resolve for it to work).

Since the final presentation output breaks the content of your XML document into many files (one for each slide), to see this output you will have to run a batch process from a tool or command line, rather than a client-side transformation.

## ***The slideshow document model***

### **Dates, versions, and the change log**

Elements for tracking the creation and change history of a slideshow are provided in the **<history>** element of the **<control>** element. For a one-off slideshow, you may not need these; but they may be useful if you rework a presentation given on an earlier occasion.

The title page of the printed slide show, and the front page of the web sequence, will display the date and version of the first **<change>** element appearing. Thus it is important that changes be logged in *reverse chronological order* (the most recent first). If there is no **<change>** element, the date and version of the **<creation>** element will be used. (*NOTE:* in the [prototype] XSLT 1.1 processing, the **<change>** or **<creation>** element with the *highest version number* is the element used, irrespective of its place in the sequence.)

Within both **<creation>** and **<change>**, **<date>** is required and **<version>** is optional. Include a version number, but not the word “version”, which will be provided by the style sheet.

**<change>** also includes an optional **<name>** element following the date and version, where you may place your initials or another identifier.

Inside either **<creation>** or **<change>**, **<para>** or **<list>** elements may be used to describe how and for whom the slideshow has been created, its sources, the nature and extent of changes, etc. If you want very detailed tracking of changes, however, it is best to check the document into a version control system.

### **The structure of a slideshow**

Mulberry’s specialized (not to say peculiar) requirements are such that we have developed an “elastic” document model for the slideshow, providing several options for managing and presenting the content in an organized way. The intention is to make it possible to offer a streamlined presentation with no extra organizational overhead (in the form of headers, directories, hierarchies etc. not really useful to the audience), while nevertheless supporting as much hierarchical organization as the creator wants, either visible or implicit, thereby encouraging rationality, intelligibility and portability of the content created, as well as making printed materials as useful as possible for referencing.

This nested segment model is what distinguishes this design most from other presentation technologies we have seen or worked with, which assume either that presentation is flat, so internal organization is flat, or that any internal hierarchy used to organize and manage material, should be exposed in output. Our design allows you to organize materials into hierarchies, but not display it that way if you choose not to.

### ***The segment model and slideshow organization***

All information in a slideshow projected on screen to an audience appears in slides (or *headslides*, a special type of slide). Slides may be gathered together in segments for organizational purposes.

Segments may contain lower-level segments as well as slides. This recursive structure allows your segment hierarchy to be as deep as you wish. Slides may appear alongside of, or within, segments at any level. In a segment, that is, slides may appear before sub-segments, in between, or after them, to provide for a review or wrap-up.

Other elements, namely exhibits, exercises and appendixes (discussed below), can also appear intermixed with slides. These provide ways of including content to be printed in a handout accompanying the slideshow, without its appearing on slides.

It is possible, if you prefer, to organize your material into segments without that organization being directly represented to the viewer. (See the next section.)

1. Use a **<segment>** element whenever you want to relate slides, and/or lower-level segments, together in a sequence (whether or not you want that relation to be exposed to viewers – see below).
2. Use a **<slide>** (or **<headslide>**) element when you want a slide to be presented.

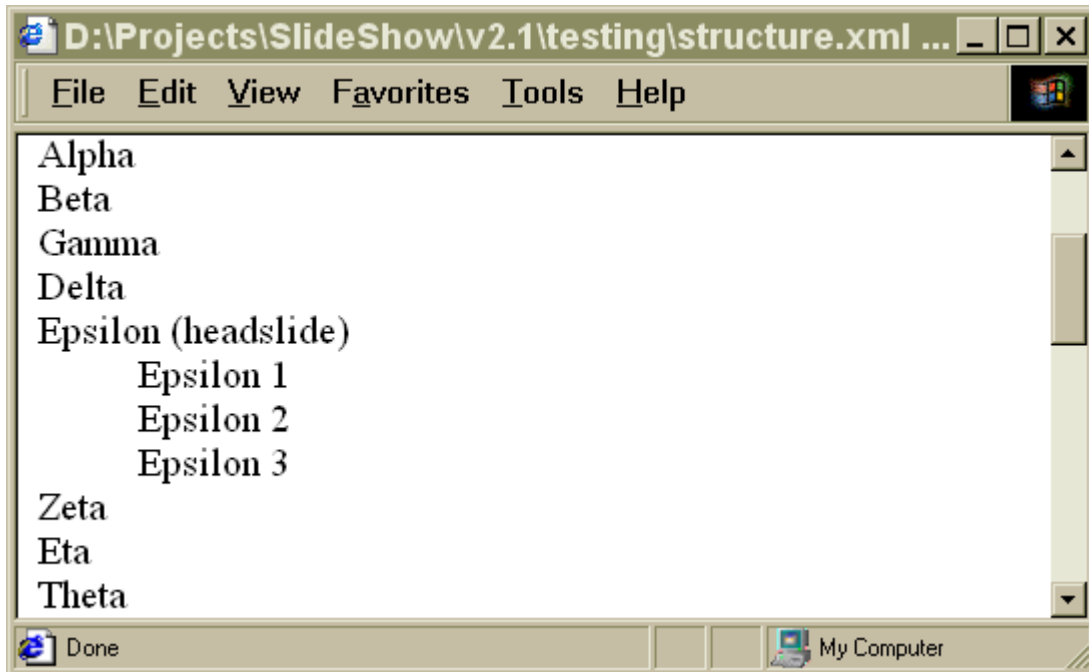
### ***Different ways to start segments reveal or hide the slideshow structure***

The first element in a segment must either be a **<headslide>**, a **<segmenthead>** or a **<ghosthead>**. (If you choose **<segmenthead>**, you may precede it with a **<tochead>**; see below.) These are all alike in that they each provide the segment with a title. The difference is that a headslide produces a slide (the element is exactly like a normal slide in all respects except its position in the segment), and the segment's title is considered to be the title of the headslide (which appears displayed accordingly in the table of contents); whereas the others give ways of giving titles to segments without presenting that event as such ("here we start a new segment!") in the sequence of slides presented.

This range of choices reflects two different strategies to giving a presentation, plus an intermediate alternative:

1. Expose the hierarchy by using "head" slides (**<headslide>** elements). When a new segment starts, the audience will see a slide to provide a kick-off. The slide will be represented differently in the print (more prominently) than normal slides, and will be listed with subsequent slides from its segment nested (indented) *under* it in the table of contents. Otherwise it will appear just like any other slide, and can have any content.
2. Hide the hierarchy by using ghosted heads (**<ghosthead>** elements). No one will know the segments are there except you, when you are editing the slideshow. In this case, the slideshow can be segmented internally while appearing as an uninterrupted, flat sequence.
3. Use segment headers (**<segmenthead>** elements). These will appear as headers in the print table of contents and the printed sequence, but will not appear as slides. This is an in-between way of doing it: it indicates the presentation's organization to readers of the print without announcing it in slides.

Each of these approaches results in a different type of presentation of your structured material in the print handout and table of contents (both in print and in the HTML file directory). See the figures below for examples. The three approaches can be mixed and matched as you like: each segment can be different, so you can decide in each case how you want it to be presented.



**Headslide option: structural view**

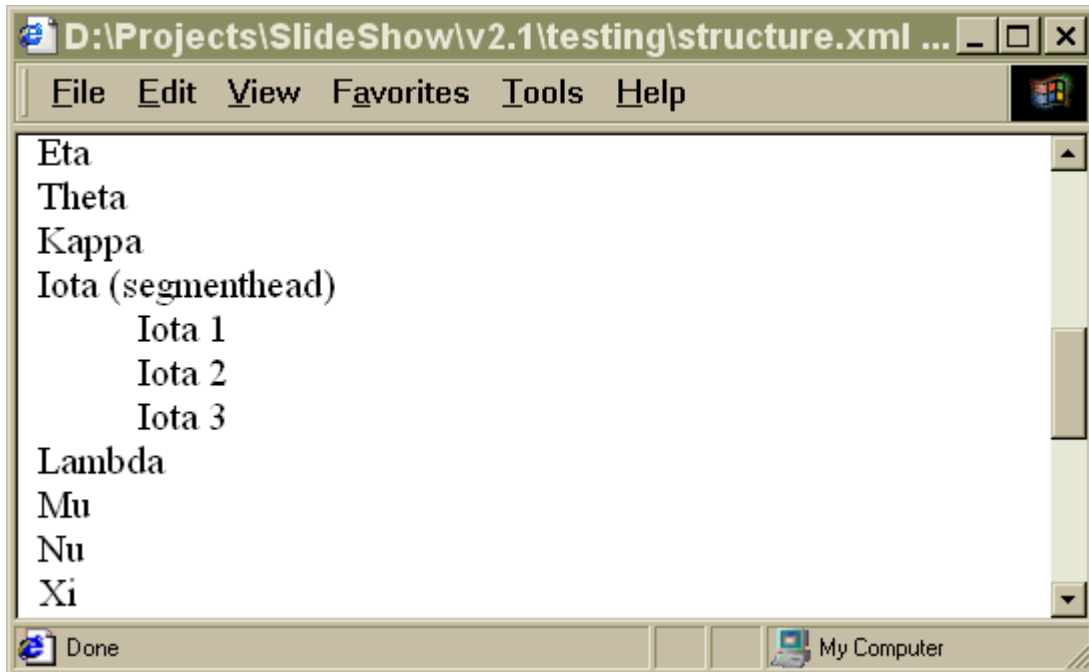
*Epsilon is a headslide; the slides Epsilon 1-3 appear together with it in a segment.  
This is a structural view.*



**Headslide option: presentation view**

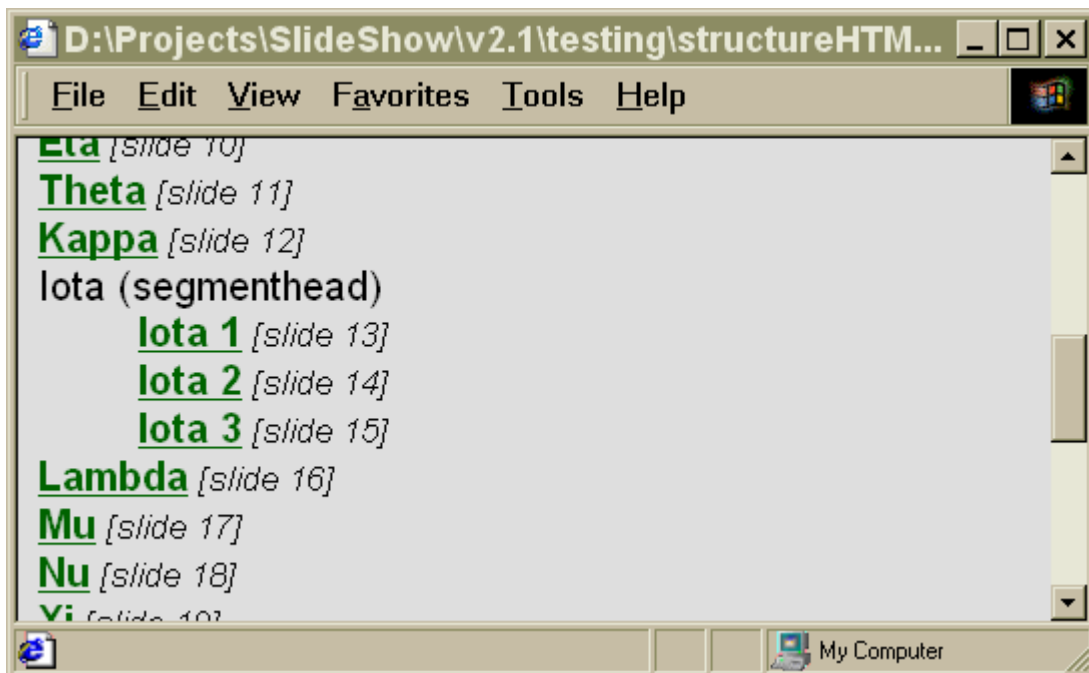
*In the table of contents, the segment structure is reflected the same way, with Epsilon 1-3 subordinate.  
Since Epsilon is a slide, it is linked (and has a slide number).*





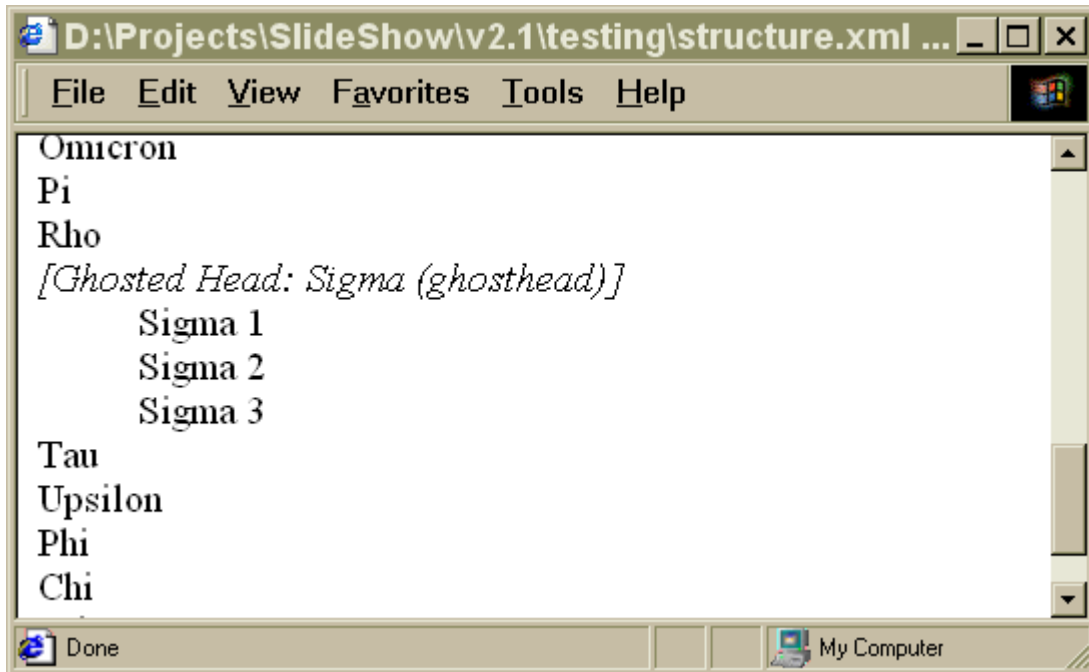
**Segmenthead option: structural view**

*Iota is a segmenthead; slides Iota 1-3 are in its segment. Lambda is not.  
(Oops! Kappa should appear after Iota, shouldn't it? well, it isn't in the segment either.)*



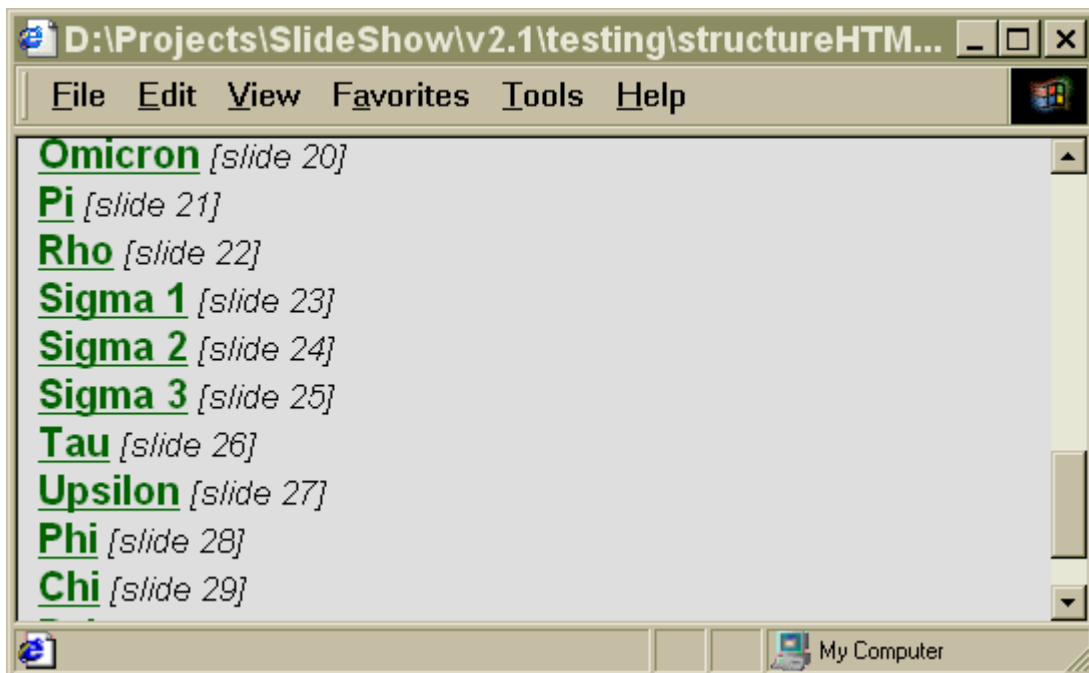
**Segmenthead option: presentation view**

*In the HTML table of contents, Iota appears without a link or slide number, because there is no slide there. But slides Iota 1-3 appear below it in the hierarchy.*



**Ghosthead option: structural view**

*Sigma is a ghosthead starting a segment, so the structural view shows it ghosted out. Slides Sigma 1-3 are subordinate to it in the actual hierarchy.*



**Ghosthead option: presentation view**

*But in the table of contents, slides Sigma 1-3 appear as if they were at the same level as the surrounding slides. Sigma does not appear and the segment is invisible.*

## ***Other options for controlling the table of contents***

### Suppressing a slide from appearing in the table of contents

Some slides should not appear in the table of contents at all. For example, if a slide is headed “Same As Last Slide (cont’d)” you may not want it to appear. A **showintoc** attribute is provided on slides; set it to “no,” and the slide will not be listed in the table.

This attribute is also provided on **<headslide>** elements, but should almost never be used there. Keep in mind that if a headslide is suppressed from the table of contents, any slides in its segment which do appear in the ToC will appear listed, incorrectly, as if they were subordinate to the most recent slide appearing.

### What does <tohead> do?

Before **<segmenthead>** elements in segments and before **<head>** elements in slides and headslides, an optional **<tohead>** element may be included. This “table of contents” head is an alternative head to be used in a table of contents in place of the regular head: use it if the regular head is too long to fit well within the table of contents, or if you want a different header there for some other reason.

## ***An XSL stylesheet for viewing an outline***

If you are not using a structured editor (such as SoftQuad XmetaL) that can show you an “outline view”, it is very convenient to be able to see a document’s actual structure (as opposed to its apparent structure, in which segments with ghostheads are flattened).

In this distribution is an XSLT stylesheet to provide such a view: `showstructure.xsl`. It will show all segments, and any exercises, exhibits and appendixes you may have. To use it, include a processing instruction after the XML declaration in your slideshow as follows:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="XSL/showstructure.xsl"?>
```

Then process your document with an XSLT engine (Internet Explorer works fine if installed with MSXML3).

## **Notes in slideshows**

Another element, **<notes>**, is permitted within segments or within individual slides (after the **<body>**). The content of this element can be printed in a proof version of a slideshow, for review, but will not be printed in a final version to be distributed.

## **Exhibits, exercises and appendixes**

These three elements are provided for cases where materials are too lengthy, or otherwise unsuitable, to be presented in a slide: this is very useful if, for example, you are teaching a course from a slideshow and need to create a handout. The stylesheets will give each of these elements a number. The three element types are numbered in three separate sequences.

- Use an **<exercise>** element when you are creating a handout for an exercise. Exercises will appear with generated text prepended, “Exercise X” (where X is the number of the exercise in the current instance), so do *not* number exercises. (Naming them, however, is nice.) Exercises will be printed at the end of the course, no matter where in the course sequence they appear. Because the numbering of exercises depends on their order and is not hard coded, you can move and rearrange exercises freely without disrupting the numbering (this also goes for **<exhibit>** and **<appendix>**).
- Use an **<exhibit>** element when you have a longer text you want to present to your audience. This element is provided because it is sometimes better to refer the audience to a printed text than it is to try to display a long text on screen. Exhibits are numbered, like exercises; but unlike exercises, they are printed in sequence, not at the end or separately. If you want an exhibit to be printed out of sequence, use an Appendix.

Like **<exercise>** elements, exhibits are provided with a generated header, “Exhibit X,” in addition to their own **<exhibname>**, which is optional.

- **<appendix>** elements are identical to exhibits (their names are even **<exhibname>** elements), except they are numbered in their own sequence, and are printed at the end of the slideshow. Use them for general reference information for the slideshow as a whole, as opposed to information that should come in a particular place (an exhibit).

## Cross-referencing

Any slide, headslide, exercise, exhibit or appendix can be referred to from elsewhere in the slideshow. To do this, assign a value to the **called** attribute on the element you wish to refer to. **called** is an ID attribute, so each one must be unique within the slideshow instance.

Two separate functionalities for cross-referencing are supported in the slideshow. First, there is a means to generate a text for a cross-reference. For example, you may want to refer to a slide, or an Exercise or Exhibit, by number (so it can easily be found). Use the **<genref>** element for this. Second, there is a way to make any string of text into a link pointing to a slide: use the **<ref>** element for this. You can use either of these elements together or separately. The **<ref>** element does not have any effect on print, only on web output, but text generated with **<genref>** appears in both output formats.

### Generating a reference string with **<genref>**

Anywhere within slideshow text, you may refer to a different slide or exhibit with the **<genref>** (“generated reference”) element. This is an empty element with an attribute, **pointsto**, which refers to another element by its **called** attribute. The stylesheet then calculates where the referenced element is, checks its element type, and generates a reference accordingly. If the referenced element is a **<slide>** or **<headslide>**, the generated string will be “slide xxx” (where xxx is the number of the slide; note “slide” is in lower case). If it is an **<exhibit>**, **<exercise>** or **<appendix>**, it will be “Exhibit XX” (or “Exercise” or “Appendix”).

So, the code

```
<item>Practice blowing large bubbles in <genref  
pointsto="bubblesexr"/>.</item>
```

will, if somewhere in the document there is an **<exercise>** element with the **called** attribute set to **bubblesexr**, and it is the fourth exercise in the slideshow, generate the text on output:

**Practice blowing large bubbles in Exercise 4.**

### Generating a hyperlink with **<ref>**

To create a hypertext pointer from a slide to another slide, use **<ref>**. **<ref>** works like an HTML anchor: any element content becomes a hyperlink. The attribute **linksto** identifies the slide that will serve as the target of the link. Links created are one-way: place a **<ref>** element at the other end if you want to link back.

**linksto** is an IDREF attribute, so it must point to a unique ID (it will be a **called** attribute). Some caution is in order: it is possible that the target is not a slide—it might, for example, resolve as an **<exercise>** element. In this case the reference would have no working target, because exercises are not created as slides for display. Make sure your **<ref>** elements point to **<slide>** or **<headslide>** elements.

**<ref>** elements will not be rendered specially in the print version; hypertext links will appear in the HTML without any special formatting or behavior on the print side. Typically, **<ref>** will be used in combination with **<genref>** (see above).

## Regular, overview and “deep” slides

Both **<slide>** and **<headslide>** elements have an attribute, **type**, which may be set either to **content** (the default), **overview**, or **deep**. This attribute supports “flagging” of slides so that a presenter may know whether a slide should be covered in detail (in the case of overviews – slides that describe material still to be covered, or already covered – or of “deep” material, there are times when it should not).

In particular, note that there is no formal relationship between overview slides and headslides. Overview slides may occur any place in a slide show, not just at the start of a segment (for example, they may be used to present a wrapup). Then too, the headslide of a segment may present important information besides just announcing a new segment and saying what's in it, in which case it may not be an overview slide.

Assign the value `deep` if the content of the slide is included for completeness, but you wish to signal the presenter not to present it in full detail, since it can be considered deep or obscure for the purposes of the presentation.

Because both “deep” and “overview” slides are not to be presented in detail, they are both signalled on screen with an underlined title. (They do not look different in print from other slides.) Additionally, overview slides may have different background color.

## **<emph> and its attributes**

Three different effects can be designated on the inline **<emph>** (emphasis) element, using four attributes.

The default attribute values are set so that an **<emph>** element with no attributes assigned will be displayed in italics.

### ***slant***

One attribute, **slant**, controls the slant of type. It has two possible values, `italic` and `roman`. The default is `italic`.

### ***weight***

One attribute, **weight**, can be used to affect whether the element content appears in bold or normal type. Possible values are `normal` and `bold`; the default is `normal`.

Since code segments appearing in the HTML may already be displayed in bold (for better legibility), setting **weight** to “bold” will also make the on-screen display change color (by default, to green).

### ***color***

There are two attributes provided to indicate how the content of an **<emph>** element may be colored. Both these attributes are optional and have no default value, so if neither is set, there will be no color change.

### **rankedcolor**

This attribute must be one of a list of enumerated values, `rank-1` through `rank-10`. The stylesheet assigns a different color to each of these values. The colors may be tweaked in the stylesheet or in a CSS overlay for different formats.

### **fixedcolor**

This attribute can have any CDATA value. Its value will be passed through the stylesheet to the HTML for rendering, so it can be any legal HTML color value, either a three- or six-digit RGB value such as `#D0A6A6` (a kind of pink?) or an HTML named color such as `red` or `seashell`. Use at your own risk (i.e., be aware of what your browser supports).

The **rankedcolor** attribute should be used in the general case, when the particular color assigned is not important, and the main requirement is that colors be distinct. If slideshow formats or color schemes ever change, **rankedcolor** is a more dependable assignment. Use **fixedcolor** if you need a color whose value is stable and predictable, particularly if the fix is immediate (“all out-of-date text appears in red!”).

If both color attributes are assigned, **fixedcolor** takes precedence. Setting either of these colors to any value will make element content appear in bold in the print output (but not otherwise colored).

## Specialized inline elements

Brief descriptions of the elements available in line (e.g. inside `<para>` or list `<item>` elements) follow.

### ***code and codeblock***

Use `<code>` if you have a segment of text in line that you want formatted as code (it will be a fixed-space font such as Courier with size adjusted).

For blocks of code appearing as examples or illustrations, use `<codeblock>`. `<codeblock>` may be used inside list items and paragraphs, but will create a display block instead of merely providing a format to the text in line.

### ***genref***

For creating generated text referring to another part of the presentation (a slide, exhibit or exercise). See above, “Cross-referencing.”

### ***quoted***

For identifying a term or string to appear in quotes. Saves having to insert entities (but feel free).

### ***ref***

For providing a hypertext link to another slide, in web output. See above, “Cross-referencing.”

### ***resource***

The `<resource>` element is used to track resources that belong to the presentation or course. In most cases these will be names of files used in exercises, for example stylesheets. Other resources could be application programs, etc.

The only function of the `<resource>` element is to provide for simple indexing. A stylesheet or filter may be applied to list all the resources referred to by a given slideshow. This can be very useful for final assembly of course materials, so consistent use of this element is encouraged in a slideshow of any size (or one that will be reused and maintained over time).

`<resource>` will be rendered like `<code>` (that is, in a fixed-space font) on the assumption that a resource will generally be the name of a file.

### ***specref***

`<specref>` stands for “specification reference”. The element provides a way of including references to relevant specifications and standards in the print version unobtrusively, without their appearing at all in the screen display (HTML) version. In the print, contents of `<specref>` elements appear in relatively small type, surrounded by brackets. They are there only for reference purposes: do not mark up anything with this element that you plan to discuss, because it won’t appear in the screen display.

### ***term***

Although mainly used as the first part of a glossary listing, `<term>` is also available in line for marking up technical terms. Its main use is to be distinguished typographically (in bold face) from surrounding text, although terms marked up consistently could be indexed.

### ***unimp***

Any text that is included for completeness, but which the writer expects to be skipped or passed over, can be marked `<unimp>`, which thus serves as an inline alternative to the slide type “deep” or “overview” options.

Text marked as **<unimp>** will appear in grey in the screen display, and in a reduced font size in the print.

## **url**

The **<url>** element appears so that references to web pages can be made into hyperlinks in the HTML version, giving a presenter (or reader) the opportunity to browse directly to the site if the browser has a net connection.

The **<url>** element has no attributes; instead, it is the *content* of the element that gets made into the target of the hyperlink. (So **<url>http://www.mulberrytech.com</url>** is rendered as HTML **<a href="http://www.mulberrytech.com">http://www.mulberrytech.com</a>**. This design is meant to ensure that any URLs that are linked will appear explicitly both in print and in the screen display.

In the print version, contents of an **<url>** element will appear in bold type.

## **worktitle**

Use the **<worktitle>** element when you wish to identify the title of a work referred to. Its **render** attribute may be set to affect the rendition (its default value is **bold**).

## **Graphics in Slideshows**

Graphics in slideshows must be in a web-compatible format, **.gif** or **.jpg**, for HTML display. Additionally, the slideshow allows you to specify alternative graphics for printing. That way you can have a **.gif** or **.jpg** for screen display, and a vector format such as **.eps** for printing. It is your responsibility to make sure the same picture appears in both cases.

Assign the **.gif** or **.jpg** file to the **www** attribute of the **<graphic>** element. Assign the print version to the **print** attribute. If no **print** attribute value is assigned, the printed version will use the file named on the **www** attribute. The **www** attribute value is required.

## **Going Into Production**

### **Fixing up the graphics**

We have provided graphics for link buttons and branding; but they will probably not be suitable for your presentation. In particular, you should provide your own graphics for branding rather than using the very generic logos we found on the Internet:

- **arrowback.gif**, **arrowforward.gif**, **arrowup.gif**: these are button icons used on the HTML pages for navigation. Replace with your own snazzier buttons. If you use these names, you can use the same HTML stylesheets without changing them. (Make sure your buttons are placed in the icons subdirectory if you use the batch file to invoke processing; otherwise just place them with your HTML output.)
- **footer-logo.gif** and **page-logo.gif**. These are logos for branding your presentation. The images provided are merely placeholders: replace them with your own. **page-logo.gif** is used only on the front page of the print output, while **footer-logo.gif** is used by stylesheets for both output formats.

Note that the XSLT prototype stylesheet implements tag structures provided by the document model (in the **<brands>** and **<buttons>** elements in **<control>**) for overriding these default file names on a per-document basis. These will also let you have an entire string of logos in your web output if you want.

## **The Jade engine**

At Mulberry Technologies, we have quite happily used James Clark's JADE engine for slideshow processing. The DSSSL stylesheets in this package have been designed for, and tested with, Jade 1.2.1. It may be obtained from [www.jclark.com](http://www.jclark.com); subsequent versions of the software (untested with this application, but they should work) may be found at [openjade.sourceforge.net](http://openjade.sourceforge.net).

Note: if you are using Jade, the output encoding of your process may not be the same as the setting in your web browser. If you have any strange rendition of characters (particularly characters in “upper ASCII”), try changing the encoding setting in your browser (you will typically find it under the View menu).

## Using XSLT instead

Also included is a prototype XSLT stylesheet for creating HTML file output, `slides.web.xsl`. Note that this stylesheet conforms to the December, 2000 draft specification for XSLT 1.1, since it makes use of the `xsl:document` instruction (not compatible with XSLT 1.0) to create its file output. It has been tested and runs in Saxon 6.2.2, which implements `xsl:document`; the stylesheet can also be fairly easily tweaked to support extensions in other processors that provide other means to do the same thing.

Please note that at this time, the only way we provide of getting directly to a print format is through DSSSL.

## Batch file processing

In this distribution, we have included two batch files that include the necessary Jade command lines: this will simplify processing at least for those users on Windows systems. Please note that *Jade must be set up and available on your system path in order for these to run*. (Change the batch files if you wish to point directly at a copy of Jade.)

### For RTF:

Place your valid XML document at the top level of the subdirectory structure in which we have packaged the files (that is, the same subdirectory with the batch files and with this guide), and use the command:

```
jade2rtf yourslideshow.xml
```

where *yourslideshow.xml* is (you guessed it) the name of your file.

Users who are comfortable with DSSSL may be interested in a range of processing options (such as printing exercises only; printing slides without exercises, etc.) available from this stylesheet, that can be controlled with command-line arguments to Jade. Look into `printshow.dsl` to see what these are. (These are not supported from the batch file.)

### For HTML:

Place your valid XML document at the top level and use the command:

```
jade2html yourslideshow.xml.
```

This routine will create an HTML subdirectory (if one does not already exist), delete its current contents, copy into it the graphics icons in the `icons` subdirectory, and create a sequence of HTML files from your slideshow document.

## Manual finishing

Fortunately, manual finishing should be pretty much limited to applying a template (if you have and want one), and checking out page breaks in the RTF output before printing or producing a PDF file. Unless you want to do something fancy with colors, no manual finishing should be required for the HTML display.

### A “finish” template

Finishing touches to the slideshow layout may be applied by using a template in a system that renders and prints Jade’s RTF output. At Mulberry, we use such a template to apply ruled lines to slides, helping readers distinguish where they start.

### Checking and tweaking

Check the document through before printing or publishing. Are all graphics included? Are page breaks in the correct places?



Because of limitations to the RTF format and what Jade can do to control blocks, there are particular weaknesses in page breaks. Two special cases have been identified; these should account for most if not all of the manual tweaks necessary for attractive pagination:

- Where tables appear as the last item in a slide, pagination may be disrupted from that point forward. This may be fixed by turning `KEEP WITH NEXT` (or its equivalent) *off* for the last row in the table.

Pagination will fall into place for slides following if you do this in all tables appearing at the end of slides.

- Sometimes long exhibits, especially with long sections of code, will break poorly across pages. This can be fixed by any or all of three methods, in order of preference:
  1. Remove the `KEEP WITH NEXT` between significant blocks of text.
  2. Inside long code blocks, turn off the `KEEP TOGETHER` option.
  3. In the case of some exhibits, you may also want to place page breaks in by hand (in which case watch the spacing above paragraphs at the start of pages, and reduce to 0 where necessary).
- You may also need to check and update links, such as those indicating page numbers in the Table of Contents.

## Customizing colors

An intrepid user may always customize the output by editing the stylesheets.

If you don't want to go to this trouble, however, colors, fonts and so forth may be edited for the HTML output by applying a CSS stylesheet.

A sample CSS stylesheet, `slideshow.css`, is included in the distribution (in the `icons` subdirectory, to keep it out of the way). Copy this file into the subdirectory when your HTML output appears (or, if browsing the XML file directly, into the same subdirectory as your source file), and see what happens to the slides in the browser. Edit this file to change its settings. Keep in mind, of course, that CSS support in browsers tends to be somewhat spotty.