The Arboreal Docspecs System
Version 1.1

Malcolm D. Hyman

July 8, 2003

1 Introduction

This manual describes the docspecs system, as implemented in Arboreal 4.0.

The docspecs system allows Arboreal to be extended to support new XML document types. A special XML file, called a *docspecs* file, contains information about XML document types. This information specifies how Arboreal handles containers, word tagging, metadata extraction, and tag rendering. Arboreal comes with a built-in docspecs file, which provides support for the Archimedes, CDLI, termlist, and Rome (the Arboreal outline format) documents. Minimal support is also provided for XHTML. The built-in docspecs can be accessed from within Arboreal via the "magic" URL:

```
    arboreal:skeletons/docspecs.xml
```

Arboreal will also read a docspecs file that is supplied by the user. This file must be called `docspecs.xml` and placed in the user's `arboreal` directory.[1] Information about all new document types should be put into this single file. Document types defined in the user-supplied docspecs will override those in the default docspecs.

[1]The `arboreal` directory is located in the user's home directory and will be created automatically by Arboreal if it does not already exist. On single-user versions of Windows, `C:\WINDOWS` is considered the home directory. The `arboreal` directory may be renamed `.arboreal`, if the user desires.

2 Gross Structure of a Docspecs File

The root element of the docspecs file is `<docspecs>`. Any number of `<doctype>` elements may appear as children of `<docspecs>`. Each defines a single document type. The `<doctype>` element must have either a `name` attribute or both a `name` and an `ns` attribute. The values of these attributes allow Arboreal to select the appropriate doctype for a particular document when it is loaded. The `name` attribute is used to match the document's XML doctype, and the `ns` attribute is used to match a namespace URI. If the `ns` attribute is not given, then the namespace is not tested. If `name` is set to the null string, then any document whose default namespace is the same as the value of the `ns` attribute is matched. If `name` is set to the null string and `ns` is not defined, then *any* document is matched. If a document matches a doctype with a non-null `name`, then Arboreal uses that doctype. Null-named doctypes serve as defaults. The default docspecs file for Arboreal defines the default doctype as the Archimedes doctype.

Example 1: `<doctype name="foo">` matches any document with the doctype *foo.*

Example 2: `<doctype name="" ns="http://foo.bar.edu/1">` will match any document in the specified namespace.

Example 3: `<doctype name="">` matches any document not matched by a more specific doctype.

If the doctype is not specified at the start of an XML document, then Arboreal tests the name of the root element against `<name>` instead.

The `<doctype>` element contains the following children:

- `<containers>`
- `<subcontainers>`
- `<word-separators>`
- `<grapheme-separators>`
- `<langspec>`
- `<metadata>`
- `<page-image>`
- `<defs>`

- `<tree-view>`
- `<content-view>`

All these elements are required, and they must appear in order given above. Each element is dealt with in a subsequent section of this document.

3 `<containers>`

The `<containers>` section is used to define all the elements in the target doctype that should be considered as containers. All the text nodes under a container will form a single amalgamation in Arboreal. A container is restricted to a single language; the subcontainer mechanism allows for language embedding.

Under `<containers>`, any number of `<element>` nodes are allowed. An `<element>` node is empty and has the single required attribute `name`.

Example 1: To define the element `<s>` as a container, use:

```
<containers><element name="s"/></containers>
```

4 `<subcontainers>`

Subcontainers allow text in a different language to be embedded within a container. The `<subcontainers>` element has the same content model as `<containers>`. In the Archimedes doctype, `<foreign>` is defined as a subcontainer.

5 `<word-separators>`

If one or more tags are used to mark words in the target doctype, these tags should be enumerated under `<word-separators>`. If words are delimited by orthographic means such as whitespace, then this section should be left empty. The `<word-separators>` element has the same content model as `<containers>`. In the CDLI doctype, `<w>` is defined as a word-separator.

6 <grapheme-separators>

If one or more tags are used in the target doctype to enclose arbitrary content that is treated as a single character, the tags should be enumerated under `<grapheme-separators>`. The `<grapheme-separators>` element has the same content model as `<containers>`. In the CDLI doctype, `<g>` is defined as a grapheme-separator.

7 <langspec>

The `<langspec>` element provides Arboreal with the information needed to determine the language of the amalgamated text within a container or subcontainer. This element has a single child, `<attribute>`; the `name` attribute of `<attribute>` defines the attribute used to encode language in the target doctype. Values for the specified attribute must be ISO 639 language codes. A node in the target document inherits the language property of its parent node, unless the language attribute is present. If the language of the root element of the target document is not specified by a language attribute, the language of the root element is considered to be the default document language specified under `<metadata>` (q. v.). Many XML documents will use the standard `lang` attribute in the `xml` namespace for this purpose. If attributes are not used to encode language in the target doctype, supply the null string as the value of `name`.

Example 1: To define the standard XML language attribute, use:

```
<langspec><attribute name="xml:lang"/></langspec>
```

8 <metadata>

The `<metadata>` element has an `info` attribute, which defines the name of an element that contains metadata for the target document. Text within the specified element is *not* considered to be in the default language of the target document, but rather in the default language specified in the Arboreal `properties.xml` file. If a doctype lacks a metadata section, the value of the `name` attribute should be the null string.

The `<metadata>` element contains four required children, which must appear in the specified order:

- `<author>`

- `<title>`
- `<locator>`
- `<language>`

The contents of these elements are XPath queries that return the specified document metadata properties. If a metadata item is not defined for the target doctype, the corresponding element may be left empty. If queries are not defined for the metadata properties, they assume the following default values:

author	null
title	the filename of the document
locator	the filename of the document
language	the default language specified in `properties.xml`

Example 1: Here is the `<metadata>` specification for the Archimedes doctype:

```
<metadata name="info">
<author>/archimedes/info/author/text()</author>
<title>/archimedes/info/title/text()</title>
<locator>/archimedes/info/locator/text()</locator>
<language>/archimedes/info/lang/text()</language>
</metadata>
```

9 `<page-image>`

Under `<page-image>`, an element may be defined which links to the page images displayed when **Show page image** is selected from the context menu in Arboreal. The `<page-image>` element has the same content model as `<containers>`. In the Archimedes doctype, `<pb>` is defined as the page-image element.

10 `<defs>`

The `<defs>` section permits pseudoelements to be defined. The pseudoelement facility allows an element with name x and attribute/value pair $a = b$ to be considered as if it were an element y. The `<defs>` section may contain any number of `<pseudoelement>` definitions. The syntax of a `<pseudoelement>` definition is:

```
<pseudoelement name="y">
<element name="x"/>
<attr-value name="a" value="b"/>
</pseudoelement>
```

Pseudoelements are very useful in defining rendering rules for the `<content-view>` and `<tree-view>` sections.

11 `<content-view>`

The `<content-view>` section allows rules to be defined for the rendering of elements in Arboreal's "rendered content" mode (i.e., when "Content XML view" is toggled off).[2] The rule language provides some of the functionality of CSS and XSLT. Rules allow for text to be rendered before and after the content of an element, for text content to be filtered, and for text context to be rendered in a particular style. Several types of conditions may be specified for the execution of any of these actions.

The `<content-view>` section may contain any number of rules. The ordering of rules is generally significant.

11.1 `<self>` and `<parent>`

The top-level element of a rule specifies the axis along which the primary tests for an action will be applied. Two axes are available: the *self* axis and the *parent* axis, specified respectively by the `<self>` and `<parent>` elements. The content model of these elements is indicated below. All children are *optional* and may occur only once, unless otherwise specified. Elements must occur in the order indicated:

- `<element>` (**required**)
- `<if>` (only available for `<self>`; may occur any number of times)
- `<ignore-before>`

[2]In an earlier version of the docspecs format, this section was named `<tags-off>`.

- `<ignore-after>`
- `<ignore-in-parent>`
- `<render-before>`
- `<render-after>`
- `<apply-filter>`
- `<style>`
- `<style-extra>`

The `<element>` tag tests the node on either the self or parent axis. Actions are only performed for the specified element or pseudoelement. Additional tests may be specified with `<ignore-before>`, `<ignore-after>`, and `<ignore-in-parent>`. Actions may be specified with `<render-before>`, `<render-after>`, `<apply-filter>`, `<style>`, and `<style-extra>`. The `<if>` element, which may only occur under `<self>`, and which may be repeated any number of times, allows for another method of specifying conditional actions.

11.2 Actions

11.2.1 `<render-before>` and `<render-after>`

These elements contain text that is rendered before and after the contents of an element in the target document. If we have `<x>bar</x>` in the target document, and the doctype contains the rule

```
<self>
<element name="x"/>
<render-before>foo</render-before>
</self>
```

Arboreal will display *foobar* in the "rendered content" mode.

In addition to text content, two elements are permitted within `<render-before>` and `<render-after>`. The empty `<nl>` element is a convenient way of rendering a newline (line break). The `<attribute>` element allows for the value of a named attribute to be rendered.

Example 1: The following rule renders a line break and the value of the `n` attribute, followed by a period and space, before the contents of an `<l n="`x`">` element in the target document:

```
<self>
<element name="l"/>
<render-before><nl/><attribute name="n"/>.  </render-before>
</self>
```

11.2.2 `<apply-filter>`

The `<apply-filter>` action allows all the text content beneath an element to be run through a filter. The filter is specified in Perl5 substitute syntax as the value of the `rule` attribute.

Example 1: The following rule uppercases every word in the text content beneath an `<s>` element in the target document:

```
<self>
<element name="s"/>
<apply-filter rule="s/(\w+)/\u$1/g"/>
</self>
```

11.2.3 `<style>` and `<style-extra>`

The `<style>` and `<style-extra>` actions allow text to be rendered in a particular style. `<style>` applies to the text content beneath an element, while `<style-extra>` applies to the text rendered by `<render-before>` and `<render-after>`, as well as any text rendered as the result of `<if>` conditions.

`<style>` and `<style-extra>` have the same content model. They allow any of the following children, in any order:

- `<alignment>`
- `<bold>`
- `<foreground>`

- `<italic>`
- `<left-indent>`
- `<right-indent>`
- `<subscript>`
- `<superscript>`
- `<underline>`

All these are empty elements. The attributes for each are detailed in the following table:

`<alignment>`	`value`: `left`, `right`, `centered`, or `justified`
`<bold>`	no attributes
`<foreground>`	`value`: $\# + xxxxxx$ (a 6-digit hexadecimal number)
`<italic>`	no attributes
`<left-indent>`	`value`: indent in points
`<right-indent>`	`value`: indent in points
`<subscript>`	no attributes
`<superscript>`	no attributes
`<underline>`	no attributes

Example 1: This rule will display the contents of `<i>` in italics:

```
<self>
<element name="i"/>
<style><italic/></style>
</self>
```

Example 2: This rule will indent text within `<blockquote>` on both sides:

```
<self>
<element name="blockquote"/>
<style>
<left-indent value="144"/>
<right-indent value="144"/>
</style>
</self>
```

11.3 Conditions

11.3.1 `<ignore-before>` and `<ignore-after>`

With these elements, it is possible to specify conditional rendering rules. Any number of `<element>` nodes are allowed as children. A `dist` attribute determines whether the immediately preceding or following element is tested (`dist="1"`), or the element to the left or right of the predecessor or successor (`dist="2"`). The value `either` tests both contexts. The default value for `dist` is 1.

Example 1: This rule renders a hyphen before each `<g>` tag, except when a `<w>` tag appears immediately to the left:

```
<self>
<element name="g"/>
<ignore-after><element name="w"/></ignore-after>
<render-before>-</render-before>
</self>
```

11.3.2 `<ignore-in-parent>`

This element tests the parent of the context node. It has the same content model as the other *ignore* conditions. There are no attributes.

11.4 Testing Attribute Values with `<if>`

Further conditional rendering rules can be written with the `<if>` syntax. It is possible to test multiple attribute values and to render prefix and suffix strings. These strings are nested within the text generated by `<render-before>` and `<render-after>`. `<if>` conditions may only occur beneath `<self>`, and any number of occurrences are allowed. Each `<if>` has exactly two children: an `<attr-value>` node and either a `<prefix>` or a `<suffix>` sub-action node. The content of `<prefix>` or `<suffix>` is the text to be rendered.

Example 1: This rule renders the contents of the `<sic>` tag in square brackets. If the `exclam` attribute has the value `"yes"`, an exclamation mark is rendered before the right bracket.

```
<self>
<element name="sic"/>
<if>
<attr-value name="exclam" value="yes"/>
<suffix>!</suffix>
</if>
<render-before>[</render-before>
<render-after>]</render-after>
</self>
```

12 `<tree-view>`

The `<tree-view>` section allows rules to be defined for the rendering of elements in Arboreal's "rendered tree" mode (i.e., when "Tree XML view" is toggled off). Rules are of the same sort as those in `<content-view>`.

Optionally, a `<default-node>` element may appear first within the `<tree-view>` section. This element specifies the default rendering format for a node in the tree; it is only used if it is not overridden by a more explicit rule. The content of `<default-node>` may include any combination of:

- character data
- the `<node-name>` element (which will be replaced with the name of the current node)
- the `<attribute>` element (which allows for substituting in an attribute value; e.g. `<attribute name="type">` will print the value of the `type` attribute)

The remainder of the `<tree-view>` section consists of `<self>` and `<parent>` rules, as described above for `<content-view>`. Note, however, that the tree may not contain styled text (thus any `<style>` and `<style-extra>` actions are simply ignored).