MARCH 1994

# NSL
# Technical Note TN-12

# Producing HTML Documents with Scribe

*Glenn Trewitt*

digital

**Network Systems Laboratory** 250 University Avenue Palo Alto, California 94301 USA

The Network Systems Laboratory (NSL), begun in August 1989, is a research laboratory devoted to components and tools for building and managing real-world networks. Our charter is to research and develop innovative internetworking systems. We apply what we have learned to help open strategic new markets for Digital.

Our expertise is in open systems and in big networks, especially those that cross organizational boundaries. Our interest is in building real systems for real users, in order to advance the state of the art. Sometimes we work on systems inside Digital; sometimes we work directly on large revenue projects.

Our strategy, since we are a small group, is to leverage our work by using, whenever we can, existing hardware and software systems. We do this by building on the large existing body of widely-accepted networking technologies. We like to work in partnership with other groups in Digital, including large account teams and engineering organizations.

Our deliverables are the communication of ideas to other parts of Digital by building and releasing prototype systems, consulting within Digital, publishing technical reports, and participation in external research and standards activities.

NSL is also a focal point for operating Digital's internal IP research network (CRAnet) and the Palo Alto Internet gateway. Ongoing experience with practical network operations provides an important basis for research on network management.

We publish the results of our work in a variety of journals, conferences, research reports, and technical notes. This document is a technical note. We use this form for rapid distribution of technical material. Usually this represents research in progress. Research reports are normally accounts of completed research and may include material from earlier technical notes.

You can order reports and technical notes by mail by sending a request to:

Technical Report Distribution
Digital Equipment Corporation
Network Systems Laboratory - WRL-1
250 University Avenue
Palo Alto, California 94301 USA

You can also order reports and technical notes by electronic mail or browse them on the World Wide Web. Use one of the following addresses:

Internet mail:
`NSL-Techreports@pa.dec.com`

World Wide Web (outside of Digital):
`http://www.research.digital.com/nsl/home.html`

World Wide Web (Internal):
`http://nsl.pa.dec.com/nsl/home.html`

To obtain more details on ordering by electronic mail, send a message to one of these mail addresses with the word ‘‘`help`’’ in the Subject line; you will receive detailed instructions.

# Producing HTML Documents with Scribe

**Glenn Trewitt**

**February, 1994**


## Abstract

Scribe is a document production system that supports the generation of many different types of documents on many different types of output devices. HTML (HyperText Markup Language) is the native format for documents in the World Wide Web (WWW). The web is undergoing explosive growth at this time. A major factor limiting this growth is the effort required to produce high-quality HTML documents.

This note describes a Scribe device type and document type that can be used to generate HTML files from a conventional Scribe document, with minimal modification. We assume that the reader has some familiarity with both HTML and Scribe.




digital **Network Systems Laboratory** 250 University Avenue Palo Alto, California 94301 USA

# Table of Contents

The software described here is available by anonymous FTP from the Network Systems Laboratory, **nsl.pa.dec.com** (Digital internal only), in the /pub/www directory. Outside of Digital, it is available from **gatekeeper.dec.com** in the /pub/DEC/NSL/www directory.

## 1. Scribe *vs.* HTML

HTML documents are usually short, typically no more than a few text pages. Each document is named by a Universal Resource Locator (URL) that looks like:

http://host.domain/path/file

A document can contain pointers to other documents by associating its URL with some text (called the anchor) in the document. When the user clicks on the anchor, the corresponding document is retrieved and displayed.

The World Wide Web consists of the entire collection of such documents, connected by hypertext links.

A typical Scribe document differs from an HTML document in several ways:

- A Scribe document is usually much longer than an HTML document. An HTML equivalent would consist of several small documents, connected by hypertext links.
- Scribe documents are usually read in a linear order, one section after another. HTML documents are often browsed in random order. It is easy to get lost.
- The only indexing information in a Scribe document is the table of contents and, possibly, the index. HTML documents are usually very richly cross-referenced. Even the *Joy of Cooking* pales by comparison.

There are two levels of capability for producing HTML documents. The simplest just produces an HTML version of the manuscript file. In addition, with a little work, a structured set of HTML documents can be produced from the manuscript file.

## 2. Making Scribe produce HTML

In the simplest case, you have a short Scribe document that you want to use to produce a single HTML file. Two things need to be done:

1. Specify **@device(HTML)** at the beginning of your document, or use the appropriate command-line switch.
2. If your document uses **@description**, you must post-process the .html output file with the *awk* script **html-post.awk**[1]. This adds the HTML **<DD>** tags inside the description body. If you omit this step, your @description environments will be incorrectly formatted.

Not all Scribe environments can be successfully translated into HTML. The following limitations exist:

[1]See Section 11.

- @display, @example, @fileexample, @programexample, @format are all formatted using the HTML <PRE> tag (preformatted text). This is usually displayed in a typewriter font.
- HTML doesn't support centered text. @center just puts the text on a line by itself.
- Footnotes are placed in-line, enclosed in [square brackets].
- Tabbing and other format control commands don't have analogs in HTML, so these commands will have no effect, except in unfilled environments, which are formatted with <PRE>.
- The **@*** command has no effect for the HTML device, except in unfilled environments. As an alternative, the HTML library (Section 3) defines the **@br** command, which causes a line break for all device types. You should use **@br** in place of **@***.
- With the common availability of laser printers, many document types, and some Scribe users, specify the **size** environment attribute. Since HTML doesn't directly support arbitrary character sizes, specification of **size** will result in the error message:

  ```
  Output device html cannot achieve scaling.
  ```

  This is not harmful and may be ignored.

The HTML device type provides the basic formatting into HTML. The features described in the remaining sections are only available by including the library file HTML.

## 3. Making A Structured HTML Document

By including the library file:

```
@libraryfile(HTML)
```

in addition to specifying the HTML device, you can produce structured HTML documents, with inter-file links set up automatically. This command should be included after all document setup commands, such as @generate, @disable, and @style.

The resulting document is structured as follows:

- Each Chapter and Section produces a separate HTML file. The HTML <TITLE> tag is inserted automatically. If needed, you can specify "inline" Chapters and Sections that don't start new files.
- Each HTML file contains, at the very beginning, **PREVIOUS** and **NEXT** hypertext links, pointing at the adjacent sections. Chapters also contain **PREVIOUS CHAPTER** and **NEXT CHAPTER** links. These links appear as buttons.
- Each Chapter contains, at the beginning, a table of contents with hypertext links to the individual sections.

To achieve these results, the following changes must be made to the Scribe manuscript file:

- Add information to the document to construct the hypertext links. This consists of augmenting the @Chapter and @Section commands to include a *tag*, which will be used by both Scribe and HTML for cross-references.

- Add a postprocessing step that splits the Scribe-generated **.html** file into separate .html files.

The following sections describe these changes in detail.

### 3.1. Coexistence With Other Device Types

Once you've made the changes described below, you will still be able to format your document for other devices. For devices other than HTML, the HTML library contains "stub" definitions for the new commands that duplicate the behavior of the regular Scribe commands.

Summary: You can do all of the fancy HTML formatting you want, and you'll still be able to format for other devices, without changing your document.

### 3.2. Convert @Section to @MakeSection

Each @Section command should be converted to @MakeSection:

```
@MakeSection(tag=tag, title="Title of the Section")
```

This will start a new HTML file named *tag*.html at this point. If you want to continue the same file (for example, if you have several short sections), you can use the @MakeInlineSection command:

```
@MakeInlineSection(tag=tag, title="Title of the Section")
```

If you use other section-level sectioning commands, such as @PrefaceSection, you can give an additional parameter to the @MakeSection or @MakeInlineSection command:

```
@MakeSection(tag=tag, title="Title of the Section",
        command=PrefaceSection)
```

### 3.3. Convert @Chapter to @MakeChapter

If your document doesn't contain chapters, skip this part, and make sure that you specify "nochapters" to the @MakeDocument command.

Each @Chapter command should be converted to @MakeChapter:

```
@MakeChapter(tag=tag, title="Title of the Chapter")
```

This will start a new HTML file named *tag*.html at this point. If you want to continue the same file (for example, if there is very little introductory material before the first real chapter) you can use the @MakeInlineChapter command:

```
@MakeInlineChapter(tag=tag, title="Title of the Chapter")
```

If you use other chapter-level sectioning commands, such as @Appendix or @UnNumbered, you can give an additional parameter to the @MakeChapter or @MakeInlineChapter command:

```
@MakeChapter(tag=tag, title="Title of the Chapter",
        command=UnNumbered)
```

### 3.4. Add @MakeDocument at the Beginning

@MakeDocument is very similar to @MakeChapter, except that it doesn't actually start a chapter. It does determine the name of the first HTML generated. It should be placed before any text in the .mss file.

`@MakeDocument(tag=`*tag*`, title="`*Title of the Whole Document*`")`

We use a convention that the name of the top-level document in a directory is **home.html**. So, the *tag* in the **@MakeDocument** command would be "home.html".

There are two options that may be specified in the **@MakeDocument** command:

**NoChapters** Specifies that no chapters will be used. This causes the chapter forward/back buttons to be omitted.

**NoSections** Specifies that the document will not use **@MakeSection** or **@MakeChapter** commands, so it will be a single HTML file. No forward/back buttons (either for sections or chapters) will be produced.

For example:

```
@MakeDocument(tag=home.html, title="Document Title", nochapters)
```

### 3.5. Include @generate Commands for Each Chapter

The section cross references for each chapter are stored in Scribe "generated portions," each named after the chapter tag. This is the same mechanism that Scribe uses for the tables of contents. For a document named "test.mss" with a @MakeChapter(tag=intro, ...), the chapter contents for that chapter would be stored in a file named "test.intro".

Each of these generated portions requires a command to be added to the beginning of the .mss file. For each @MakeChapter, insert a line of the form:

`@generate(`*tag*=*tag*`)`

before the `@LibraryFile(HTML)` command.

If you add a chapter and forget the @generate command, Scribe will produce some very cryptic error messages:

```
Error in BACKPLACE command found while processing the manuscript.
res-html.mss, line 123:  @backplace(resources)
The name RESOURCES should be defined but is not!

Error in BACKPLACE command.
res-html.mss, line 123:  @backplace(resources)
Portion RESOURCES cannot be backplaced.
```

### 3.6. Forced Line Breaks -- Convert @* to @br

The **@*** command has no effect for the HTML device, except in unfilled environments. As an alternative, the HTML library defines the **@br** command, which causes a line break for all device types.

You should use **@br** in place of **@***.

### 3.7. Add an @HTMLfinish Command at the End

At the very end of your .mss file, add the command:

```
@HTMLfinish()
```

If you leave out this command, you will get two error messages for the last chapter and section *tags*:

```
Error in VALUE command found while processing the manuscript.
other.mss, line 241:  @value(tag-next)
The name tag-next is used in the @Value command, but it has not
been defined with @String.
```

## 4. Imbedded Graphics

The HTML library allows the **@graphic** command to be used. The only two parameters that are used are **URL**, which specifies the URL of the graphic to display, and **ALIGN**, which can be either **TOP** or **BOTTOM**, and controls how the image is aligned with the text. Other parameters, used for other devices (such as **PostScript**), are ignored. Here are two examples:

@graphic[url="/digital/pics/logo.gif", align=top]:

@graphic[url="/digital/pics/logo.gif", align=bottom]:

Note that, for PostScript output, the **align** parameter has no effect, as is obvious here.

## 5. The First Chapter Contents

If your document contains no chapters, only sections, or if your introductory material contains sections before the first chapter, no chapter contents will be generated. If you want this, you must:

- Insert an @HTMLcontents(*tag*) command at the point you want the contents to appear.
- Add an @generate(*tag*=*tag*) at the beginning of your .mss file.

The *tag* should be the same tag that you used in the @MakeDocument command.

The chapter contents is displayed as an itemized list. Here are the contents for this document:

- Scribe *vs.* HTML
- Making Scribe produce HTML
- Making A Structured HTML Document
- The First Chapter Contents

## 6. Labels, Cross References, and Anchors

Existing Scribe commands for doing cross-references include @Label, @Tag, @Ref, and @PageRef. These references can easily be converted to hypertext links, as follows:

Replace each occurrence of @Label or @Tag with @LinkLabel or @LinkTag. If an @Label command is used to identify a chapter or section, it should be removed, because @MakeChapter and @MakeSection automatically create the label.

For each usage of @Ref, replace it with @LinkRef. This is more complicated, because you have to identify the text to be the anchor of the hypertext link (*i.e.*, the highlighted text). If you have the text

```
Chapter @ref(alice) discusses other aspects of
Alice’s journey.
```

you could convert it to:

```
@LinkRef(tag=alice, text="Chapter @ref(alice)") discusses
other aspects of Alice’s journey.
```

Both of these produce the same text, but in the HTML result, "Chapter 1" would be a hypertext link.

There is no analog to @PageRef, because HTML doesn’t have page numbers. We suggest that you replace page references with section and chapter references.

You can insert hypertext links (anchors) using the @URLref command:

```
@URLref(url="file://nsl.pa.dec.com/pub/nsl-tools",
text="/pub/nsl-tools")
```

This command puts the *text* into the document as a hypertext link to the named file. The result is:

```
<A HREF="file://nsl.pa.dec.com/pub/nsl-tools">
/pub/nsl-tools</A>
```

## 7. Chapter and Section Buttons

The chapter and section buttons are produced by including inline GIF images. The Scribe string **buttondir** names the directory where the buttons are kept. The default is "/digital/pics/icons/", which you can change permanently by editing html.lib, or in each document by specifying:

```
@string(buttondir="/pics/icons/")
```

The path name must be relative to the HTTP daemon’s DocumentRoot, since it is used to form a URL, not a file name.

The following button names are used:

- left-icon-dim.gif
- left-icon.gif
- right-icon-dim.gif
- right-icon.gif
- fast-left-icon-dim.gif
- fast-left-icon.gif
- fast-right-icon-dim.gif
- fast-right-icon.gif

The "-dim" icons are used for links that don’t exist (*i.e.*, "next" link from the last page). The "fast-" icons are used for chapter links.

## 8. Character Conversions and Raw HTML

The characters **&**, **<**, **>**, and **"** are all special to HTML, and are represented by the words **&**, **<**, **>**, and **"**. When any of these special characters are encountered in a Scribe document, they are converted to the corresponding words.

This makes it difficult to produce raw HTML, for example, when setting headers and footers. So, we provide Scribe commands that produce the commonly-used HTML constructs:

| HTML construct | Scribe command | Example | HTML result |
|---|---|---|---|
| <tag> | @htag(tag) | @tag(ISINDEX) | <ISINDEX> |
| "string" | @quot(string) | @quot(filename) | "filename" |
| &entity; | @entity(entity) | @entity(quot) | " |
| <BR> | @br | @br | <BR> |
| <HR> | @bar() | @bar() | |

---

To construct an HTML anchor, you could use:

```
@htag(A HREF=@quot(http://www.digital.com/archive/pub/DEC/DECinfo/html/home.html))
Digital’s home page@htag(/A)
```

Which would produce:

```
<A HREF="http://www.digital.com/archive/pub/DEC/DECinfo/html/home.html">
Digital’s home page</A>
```

The preferred method is be to use the @URLref command described in Section 6.

## 9. Headers, Footers, and Addresses

Most HTML documents have a footer at the bottom to indicate authorship. In addition, you may want to put some sort of a header at the beginning of each HTML file. There are two strings that you may define the tell Scribe what to insert for headers and footers: **HTMLheader** and **HTMLfooter**. The definitions that were used for this document (assuming that you are viewing it with a WWW browser) are:

```
@string(htmlheader={@majorheading(@htag(IMG
SRC=@quot(/digital/pics/logo.gif))
Producing HTML Documents with Scribe)
})

@string(HTMLfooter={@bar()
@address[Copyright (C) 1993, 1994 Digital Equipment Corporation]
@address(@urlref(url=/nsl/people/trewitt/bio.html,
text="Glenn Trewitt"), trewitt@pa.dec.com)
})
```

Note the @address command -- it marks the text with an <ADDRESS> tag.

Scribe page headings and footings are not relevant to the HTML device, because it is not paginated in the normal way. Therefore, the two mechanisms are completely independent.

## 10. The <TITLE> Tag

The HTML <TITLE> tag is automatically generated from the @MakeDocument, @MakeChapter, and @MakeSection commands, using the appropriate title as the HTML document title. If your document is going to be indexed (for example, by an automated web-walking program, then these titles probably won't make any sense outside the context of your document.

If you set the string **titleprefix**:

```
@string(titleprefix="Scribe to HTML: ")
```

then that value will be prepended to each <TITLE>. If you are viewing this report via the Web, you can see the effect of this by looking at the document title for this page. Note that the trailing space in the string is significant.

## 11. Postprocessing the .HTML file

After Scribe has produced the .html file, it must be post-processed, in order to split it apart into individual HTML files, and to add the <DD> tag for @description environments.

The following command does the actual work:

```
awk -f html-post.awk <file.html
```

All of the resulting HTML files are placed in a subdirectory named **html**, which must already exist.

Here is the actual awk script, which should be named **html-post.awk**:

```
BEGIN { file = "" }

/<P> %DOCUMENT%/ {
        if (file!="") close(file);
        file = "html/" $3 ".html";
        next;
}

file == "" { next }

#  Remove all backspaces.
{  gsub("^H", "", $0)  }

#  Add the <DD> tag to the line after the <DT> tag.
/<DT>/  {
        if (file == "") { print $0 ; printf "<DD> " }
        else { print $0 >file ; printf "<DD> " >file }
        next;
}

{
        if (file == "") print $0;
        else print $0 >file;
}
```

## 12. How the Conversion and Structuring Works

The material in this section is only interesting if you want to modify the HTML library, and need to understand how it works.

### 12.1. The HTML Device Type

This turns out to be relatively simple, with just a few tricks.

The HTML raw font is normal, except that the four special characters **&**, <, >, and **"** are redefined to emit the HTML entity codes **&**, **<**, **>**, and **"**.

To actually emit the raw characters **&**, <, >, and **"**, a raw font, HTMLsymbols, is defined that maps all of these characters directly. For most uses, the commands in Section 8 should be used. If you are desperate, you can use the @H(*char*) environment, which will emit the raw text.

Environments that map directly to HTML constructs, such as itemize, enumerate, example, etc., are handled by specifying **BeforeEntry** and **AfterExit** to insert the tags. Per-paragraph tags, such as <LI> and <DT> are put in using the **Numbered** environment attribute.

Headings and chapter title environments are handled the same way, with the <H1>, <H2>, etc. tags.

The <P> separator between paragraphs is inserted by making the **text** environment **numbered**, with the string <P>. Since both the **text** and list-making environments use the **numbered** attribute to insert tags, using **@multiple** to group multiple paragraphs into one list item is not possible.

## 12.2. HTML tags vs. Scribe Character Counting

Scribe counts each character output to the .html file as a significant character -- it thinks that the string "<B>bold</B>" is 11 characters long, even though it will be seen by the user as only 4 characters. This can severely disrupt the alignment of text in formatted environments. When anchors are involved, where dozens of characters may be "invisible", line breaks may be affected enough to make the formatting intolerable.

The solution used in the HTML device definition is that all such tags are output with *tag*, which makes Scribe output the tag, then the same number of backspace characters, so that it treats the string as having zero width. The awk script removes the backspace characters.

A better solution would have been to define a font for printing HTML tags with, that has zero-width characters. Unfortunately, Scribe didn't properly interpret this when we tried it.

## 12.3. Document Structuring and Links

The Scribe facility for generated portions is heavily used. The most important generated portion is **tags**, which contains @string definitions for cross references. A combination Scribe/HTML tag is generated for each:

- **@LinkTag** command in the document
- **@MakeDocument**
- **@MakeChapter** or **@MakeInlineChapter**
- **@MakeSection** or **@MakeInlineSection**

This combination tag consists of the following components:

- A conventional Scribe tag, generated with @Label.
- A **@string** definition of *tag*-file, which names the HTML file that the tag occurs in.
- An HTML anchor in that file, at the location of the @LinkLabel or section/chapter heading.

For chapters and sections, the Scribe tag and the HTML anchor are placed slightly differently: The Scribe tag is located after the @Section or @Chapter command, so that it will have the correct number associated with it. The HTML anchor is placed immediately before the @Section or @Chapter command, so that the heading will be visible when the link is referenced.

The following strings are defined in the **tags** generated portion:

| | |
|---|---|
| *tag*-file | HTML file that contains the tag. |
| *tag* | Name of the Chapter or Section (Only for chapter and section tags.) |
| *tag*-next | Next HTML file after this one. (Only for chapter/section tags that start a new file.) |
| *tag*-nextc | HTML file for the chapter after this one. (Only for chapter tags that start a new file.) |

For each chapter (inline or not), a generated portion is used to contain the hypertext links to the sections in that chapter. The @HTMLcontents textform does a @backplace of the portion to insert the chapter contents at that point.