26. An introduction to LAML

LAML means 'Lisp Abstracted Markup Language'. LAML is a software package written by the author of this material. The purpose of LAML is to support web development in Scheme - both of static materials and of server-side applications.

In the first and the main part of this material we have used LAML whenever possible, to illustrate functional programming in Scheme with examples from the web domain. We are now ready for a more solid introduction and description of LAML.

26.1. HTML documents in LAML

Lecture 7 - slide 2

We start gently by taking a look at a very simple HTML document - see Program 26.1. The document in Program 26.1 can be accessed via [initdoc].

> We introduce LAML by studying the LAML counterpart of a very simple HTML document

```
<html>
   <head>
     <title>This is the document title</title>
   </head>
   <body>
      <h1>Document title</h1>
      <p>Here is the first paragraph of the document</p>
      <p>The second paragraph has an <em>emphasized item</em>
         and a <b>bold face item</b>.
      </p>
   </body>
</html>
```

Program 26.1 *A very simple HTML document that illustrates the overall HTML document structure. It may, however, be tedious to write these tags for each and every small HTML page you have to produce.*

The LAML counterpart of Program 26.1 is shown in Program 26.2. Each HTML element instance used in Program 26.1 is available in Scheme as a function. The set of such functions are called a *mirror of HTML in Scheme*. In Chapter 27 we will study the HTML mirror functions in details.

```
(html
 (head
  (title "This is the document title")
 )

 (body
  (h1 "Document title")

  (p "Here is the first paragraph of the document")

  (p "The second paragraph has an" (em "emphasized item")
     "and a" (b "bold face item") _ ".")
  )
)
```

Program 26.2 *The same document as an LAML expression. We see that the the shift from HTML to LAML is a matter of a few changes, some of which can be claimed to be of lexcical nature, and some of which are of concrete syntactical of nature. Notice the use of the underscore character, which suppresses the insertion of white space. This document cannot be processed immediately. However, the next version that we show can be processed.*

The final document on this page, Program 26.3, shows a complete LAML document. The red parts of the program make up the differences between Program 26.2 and Program 26.3. The first two lines of the red parts load the necessary software. The `write-html` clause renders the HTML document to text, and it writes the HTML document to a file in the file system.

```
(load (string-append laml-dir "laml.scm"))
(style "simple-html4.01-transitional-validating")

(write-html '(raw prolog epilog)
 (html
  (head
   (title "This is the document title")
   )

  (body
   (h1 "Document title")

   (p "Here is the first paragraph of the document")

   (p "The second paragraph has an" (em "emphasized item")
      "and a" (b "bold face item")_".")
   )
  )
)
```

Program 26.3 *The document from above embedded in the LAML framework. Besides initial loading, we see the imperative writing of the functionally generated document to a specific target file. If the source file is `doc1.laml`, the target file will be `doc1.html`.*

The document generated by Program 26.3 can be accessed via [lamldoc].

26.2. Authoring of web materials

Lecture 7 - slide 4

We leave the HTML mirror functions temporarily. We come back to them in Chapter 27. Here we will give an overview of ways to produce web materials. Our main purpose with this section is to put our approach - programmatic authoring - in perspective and in relation to more well-known and more commonly used techniques.

Several different approaches to web authoring can be identified

- **Writing the document in a markup language**
  - HTML - low level and non-extensible
  - XML - requires subsequent transformations or specification of the document rendering
- **Using a visual tool - a structure editor on top of the markup language**
  - Good for low skill users
  - Difficult to manage large and complex materials
- **Transforming the document from another format**
  - Will often result in a web edition of a paper document
  - Difficult to make effective use of the WWW's hypertext potentials
- **Writing the document in a programming language**
  - Potentially good for users with programming skills
  - To be explored in this lecture

In the next section we define and describe the last approach - programmatic authoring - in more details.

26.3. Programmatic authoring

Lecture 7 - slide 5

Programmatic authoring is the idea behind the use of LAML for creation of static web pages.

Using *programmatic authoring* the document source is a program written in a programming language. By executing the program, the document source program is transformed to another format, typically HTML.

The generation of HTML upon execution of a programmatically authored document can be questioned. It can be argued that several possible kinds of processings and transformation can be imagined. Why bind the program execution to a single kind of processing?

In LAML we have more recently prepared for an intermediate document representation, which is produced when a document is processed. This is part of the XML-in-LAML approach, which we touch on in Section 28.4. This document representation is akin to abstract syntax trees. Thus, when

evaluating an expression such as the one in Program 26.2 in a context where all the mirror functions and defined, an abstract syntax is returned. This intermediate representation can be rendered as textual HTML, or it can be processed in another direction.

- Expected advantages of programmatic authoring
  - We can use all the *'programming tricks'* in the web authoring area
    - Authoring of complex materials parallels creation of non-trivial programs
  - Programmatic authoring is probably not feasible in many main stream languages
    - Java, C++, C, Pascal, Perl, ...

Using programmatic authoring the power of a programming language is available *everywhere* in the document, and at *any time* in the authoring process

26.4. LAML: Lisp Abstracted Markup Language

Lecture 7 - slide 7

We now introduce the LAML system, which basically is Scheme with access to libraries which mirror HTML. In addition, we support a number of functions which in our experience are very helpful during a typical web authoring process. Some of these functions are organized in document styles, others in tools of various kinds.

LAML means Lisp Abstracted Markup Language

LAML provides abstractions, in terms of functions, for HTML. Beyond these it is possible to create arbitrary abstractions, along the line of XML.

- LAML fundamentals:
  - The Scheme programming language itself
  - Mirrors of HTML and XML languages in Scheme
  - A number of useful libraries
  - A number of LAML document styles
  - A number of LAML tools
  - An Emacs Environment for practical use of LAML

Briefly stated, but also somewhat simplified, we can summarize LAML as follows.

LAML = Scheme + The HTML and XML mirrors

26.5. References

[lamldoc] The LAML generated document as rendered in a browser
http://www.cs.auc.dk/~normark/prog3-03/html/notes/external-html/doc1.html

[initdoc] The initial HTML document as rendered in a browser
http://www.cs.auc.dk/~normark/prog3-03/html/notes/external-html/initial-document.html

27. HTML mirror functions in LAML

In this chapter we will elaborate on the HTML mirror functions, which we have encountered numerous times during the initial parts of this material.

The mirror functions are generated automatically from a document type definition of HTML.

The apparatus for generation of HTML mirrors is also available for XML languages. This is XML-in-LAML. We have used XML-in-LAML for several non-trivial tasks. The source of this material is authored in an XML-in-LAML language called LENO.

27.1. The HTML mirrors in LAML

Lecture 7 - slide 9

We will here describe the HTML4.01 transitional and the XHTML mirrors in LAML. All the essential observations also hold for XML-in-LAML languages. The XHTML mirrors are made via the XML-in-LAML framework.

A *HTML mirror in Scheme* is a set of functions that in an accurate way makes the HTML elements of a particular HTML version available in Scheme

The key properties of a HTML mirror in Scheme - as provided by LAML - are the following.

- A one-to-one mapping of HTML elements to named functions in Scheme
- Generates **well-formed** and **valid** HTML documents
- Prevents accidental emission of '<', '>', and '&' as part of the textual contents
- The mirror functions return abstract syntax trees which can be rendered as 'HTML text'
- Supports optional pretty printing of the resulting HTML code

The generation of valid HTML in reality boils down to check of the context free composition of the document on the fly, while it is generated by the mirror functions.

As of the fall of 2003, the generation of the validation procedures is fully automatic. XML elements with so-called *element contents* [xml10] (as opposed to the elements with *mixed content*, i.e. elements with alternatives and PCDATA) are validated by deterministic finite state automata.

The point below is rather naive, but nevertheless important.

Not too many functions - not too few. You cannot by accident use a non-standard HTML element

Using raw XML you can by accident use a non-existing tag or element. When used through LAML you will discover this as soon as you attempt to execute the LAML document - simply because the counterpart of the non-existing tag is a non-defined Scheme function.

In Section 27.2 and the subsequent sections we will describe and discuss the detailed rules of the mirror functions. The rules will be summarized in Section 27.6.

27.2. Mirroring of HTML (1)

Lecture 7 - slide 10

We describe the mirror functions in this and the following sections.

We here focus on the basic elements and their composition. As a contrast to Program 26.2 from the previous chapter, we also include the attributes in the forms below

> The mirror function `f` distinguishes between *attribute names*, *attribute values*, *explicit white space*,*character references*,and *content strings* via the runtime types of the parameters together with their mutual position in the parameter list.

```
(f 'a1 "v1" 'a2 "v2" "Some text." "More text")
```

Program 27.1 *A HTML mirror expression with attribute names (symbols), attribute values (strings following symbols) and content strings.*

The LAML Scheme form in Program 27.1 corresponds to the HTML form in Program 27.2.

```
<f a1="v1" a2="v2"> Some text. More text</f>
```

Program 27.2 *The rendering of the value of the HTML mirror expression.*

Attribute names are represented as symbols. The string that follows a symbol is the attribute value. All other strings are 'white space concatenated' to the element instance contents. The details of the white space handling is described in Section 27.3.

As a consequence, and as a generalization in relation to HTML and XML, the attributes and the contents may be mixed arbitrarily. As an example, all the attributes may come after the content strings.

The HTML mirror functions traverse the actual parameters and 'sorts' them into attributes and contents contributions. From a programming language point of view, the attributes are handled as keyword parameters - although simulated with use of symbols as explained above.

27.3. Mirroring of HTML (2)

Lecture 7 - slide 11

The handling of white spaces is a minor detail. Nevertheless, it is a detail which is important to 'get right'.

The basic idea behind the handling of white spaces in LAML is formulated in the following point.

> Instead of specifying where to add white spaces we tell where to suppress it

The underscore symbols (shown with red emphasis) in Program 27.3 suppress white space. Underlying, the underscore symbol is bound to a boolean false value. This is rather arbitrary. What matters is really that a distinguished and unique run-time value is used for the purpose. Symbols and strings are ruled out, because they are used already for content and attributes. A boolean value is fine.

```
(p "Use" (kbd "HTML") _ "," (kbd "XHTML") _ ","
        (kbd "XML")_ "," "or" (kbd "LAML") _ ".")
```

Program 27.3 *An HTML mirror expression which suppresses white space in front of punctuation characters.*

The Scheme fragment in Program 27.3 returns an internal structure, which can be rendered as shown in Program 27.4.

```
<p>
  Use <kbd>HTML</kbd>, <kbd>XHTML</kbd>, <kbd>XML</kbd>, or <kbd>LAML</kbd>.
</p>
```

Program 27.4 *The rendering of the value of the HTML mirror expression.*

27.4. Mirroring of HTML (3)

Lecture 7 - slide 12

Until now we have met symbols, strings and boolean values as parameters in the HTML mirror functions. We have not yet used lists. We will do that now.

When a list is encountered among the parameters of a HTML mirror function, the elements of the list are spliced into the surrounding parameter lists. The implementation of the splicing is recursive, such that lists of any depth are flattened and spliced in the contextual list of HTML mirror function parameters.

It turns out that this handling of this is extremely flexible and important in a language, where the primary structuring of data is done with lists.

List of contents and lists of attributes are processed recursively and spliced together with their context

In Program 27.5 we see an `ul` unordered list, in which the contents is passed as a list. The list is formed by mapping the `li` mirror function on `(list( "one" "two" "three")`.

```
  (body

    (ul
      (map li (list "one" "two" "three"))
    )


    (let ((attributes
            (list 'start "3" 'css:list-style-type "lower-roman"))
          (contents (map li (list "one" "two" "three"))))
       (ol 'id "demo" contents attributes)))
```

Program 27.5 *An HTML mirror expression in which lists are passed as parameters to the HTML mirror functions.*

With the convention of splicing lists into its surround, the example would be more clumsy too, because we then have to apply `ul` on a certain list, using `apply`, cf. Section 25.4.

In the `let` construct of Program 27.5 we bind the blue names `attributes` and `contents` to two lists. Both lists are spliced into the list with the `id` symbol and the "demo" string. In the `ol` form we just refer to these lists via their names. Notice that the `id` attribute of `ol` is passed 'the normal way'.

```
<body>
    <ul><li>one</li> <li>two</li> <li>three</li></ul>

    <ol style="list-style-type: lower-roman;" id="demo" start="3">
        <li>one</li>
        <li>two</li>
        <li>three</li>
    </ol>
</body>
```

Program 27.6 *The rendering of the value of the HTML mirror expression.*

27.5. Mirroring of HTML (4)

Lecture 7 - slide 13

In this section we show how CSS attributes - Cascading Style Sheet attributes - are handled in the HTML mirror functions.

CSS attributes and HTML attributes are uniformly specified

CSS attributes are prefixed with `css:`

```
(em 'css:background-color "yellow" "Functional Programming in Scheme")
```

Program 27.7 *An HTML mirror form in which we highlight a CSS attribute.*

The main point to notice is the uniform notation used for both HTML attributes and CSS attributes. The notation is somehow in conflict with the notation used for name spaces in XML.

The rendering of the expression in Program 27.7 is shown in Program 27.8.

```
<em style="background-color: yellow;">Functional Programming in Scheme</em>
```

Program 27.8 *The rendering of the value of the HTML mirror expression.*

HTML attributes are validated in LAML, but CSS attributes are not.

27.6. Summary of Mirror Rules

Lecture 7 - slide 14

The parameter passing rules of the HTML mirror functions are summarized below. Notice that the exact same rules apply for XML mirror functions in the so-called XML-in-LAML framework.

The HTML mirror conventions of LAML can be summarized in six rules

- **Rule 1**
  An attribute name is a symbol in Scheme, which must be followed by an expression of type string, which plays the role as the attribute's value.
- **Rule 2**
  Parameters which do not follow a symbol are content elements (strings or instances of elements).
- **Rule 3**
  All content elements are implicitly separated by white space.
- **Rule 4**
  A distinguished data object (the boolean value false) which we conveniently bind to a variable named _ suppresses white space at the location where the value appears.
- **Rule 5**
  Every place an attribute or a content element is accepted we also accept a list, the elements of which are processed recursively and unfolded into the result.
- **Rule 6**
  An attribute with the name **css:** ***a*** refers to the ***a*** attribute in CSS.

In the next chapter we will discuss a number of practical aspects of the LAML system.

27.7. References

[xml10] World Wide Web Consortium, "Extensible Markup Language (XML) 1.0", February 1998.

28. Additional LAML topics

In this chapter we describe some more practical aspects of using LAML. We also touch on the XML-in-LAML approach.

28.1. LAML document processing

Lecture 7 - slide 16

As one of the underlying ideas of LAML, we organize the programmatic source of a web document as a Scheme program. When the Scheme program is executed, the underlying HTML files are generated.

It is of practical importance to make it easy and flexible to execute the Scheme program. Below we summarize the possibilities supported in the LAML system.

The LAML system supports a number of different ways to process a document

- **From the command prompt (shell)**
  - Good for some Unix users
  - Good for tool composition - piping
- **From a Scheme prompt (LAML prompt)**
  - Makes it possible to interact with LAML at a more fine grained level
- **From Emacs**
  - Direct support of LAML from an advanced text editor
  - Synchronous and asynchronous processing
  - Keyboard shortcuts, templates, and menu support
  - Support of embedding of a substring in a Lisp form - and the inverse unembedding

Let us be more concrete. Scheme source files with LAML documents are supposed to have the file extension 'laml'. We will assume that we have written source document in a file called 'd.laml'.

From a command prompt or a Unix shell, you can execute it and hereby generate the underlying HTML file by the command `laml d`.

From a Scheme interpreter (a Scheme prompt) in which the basic LAML software has been loaded, you can type `(laml "d")` followed by "enter". This is a call of the `laml` procedure on "`d`".

From within Emacs (in which the LAML mode has been loaded) you can process a the document directly, by issuing the command **M-x laml-process-current-buffer**. (It is assumed, however, that the buffer is associated with a file at some location in the file system). The emacs command `laml-process-current-buffer` is bound to some convenient shortcut, `C-o`, which makes it very easy to do the processing.

In Emacs, it is possible to start an interactive Scheme/LAML session. This is done with **M-x run-laml-interactively**. Using this possibility, the Scheme system is started, and selected LAML libraries are loaded. The Emacs Lisp variables `interactive-laml-mirror-library` and `interactive-laml-load-convenience-library` control which HTML mirror library and which additional support libraries to load in the interactive LAML session.

> The Emacs support of LAML can be seen as powerful *environment* for programmatic authoring

As a practical remark, we will strongly recommend that LAML is used via Emacs. Authors of LAML-based materials should make use the features of the Emacs laml mode. Similarly, users who interact with the Scheme system can make good use of the additional support in Emacs, such as the `run-laml-interactively` command mentioned above, and the variables that control the behavior of this command.

28.2. LAML document styles and tools

Lecture 7 - slide 17

The purpose of this section is to give a short overview of some of the major document styles and tools in LAML. A document style is a LAML based language. All such new languages are derived from an XML document type definition.

The overview below is by no means comprehensive. You are referred to the LAML Software home page [normark02e] for details.

> A number of document styles and tools have been built on top of the basic LAML libraries and the mirrors

- Document styles
  - *Domain specific web languages*
  - LENO in which this material is written
  - Course plan in which the course home page is written
  - The Manual document style in which the LAML software is documented
  - ...
- Tools
  - The Scheme Elucidator used to explain programs in this material
  - Web Calendar used on the course home page
  - XML parser and XML mirror generation tool
  - ...

28.3. LAML server-side programming - CGI

Lecture 7 - slide 18

LAML can be used for authoring of static web material using a Scheme mirror of an XML language. LAML has also been used for server side programming using the Common Gateway Interface - CGI.

In this material we will not cover CGI programming in Scheme, but we will refer to an elucidative tutorial on the topic.

In the web version of this material there is an 'Elucidate' button which brings you to a Scheme elucidator that explains how to make a LAML CGI program, cf.[laml-cgi-tutorial].

A number of non-trivial server side web systems have been made in LAML via use of CGI

- LAML Web System
  - IDAFUS - a distance education manager used for CS open university activities during three years
  - The calendar manager
  - The exercise manager
  - ...

IDAFUS - ***I****nsitut for* ***DA****talogis* ***F****jern****U****ndervisnings* ***S****ystem* - is a major example of a LAML server application used for non-trivial purposes during a number of years in open university educations. The exercise manager is a minor system for synchronous management of exercise sessions. It requires user name and password to try out these systems.

The LAML calendar manager is widely used at Aalborg University for course and semester calendar management. The author's (simple) calendar can be accessed as an example [kn-calendar]. From a given calendar you can create your own, if you want.

28.4. XML in LAML

Lecture 7 - slide 20

XML-in-LAML is the LAML framework for mirroring of XML languages in Scheme.

In LAML there is a DTD parser, which produces a Lisp representation of a DTD. Based on this representation it is possible to synthesize an exact Scheme mirror of the XML language. The properties of this mirror is described below.

XML-in-LAML is a mirroring technique that makes XML languages available in Scheme

- **XML-in-LAML**
  - Mirror derivation from Document Type Definitions (DTDs)
  - Automatic derivation of validation predicates
  - Two or more XML languages can co-exist
- **Existing XML-in-LAML languages**
  - XHTML (strict, transitional, frameset) with full validation
  - SVG - Scalable Vector Graphics - with partial validation
  - LENO - the XML language used as the source of this material - with full validation
  - Course Plan - the course home page system.
  - Program Dissection - A simple program explanation facility
  - Photo Show - A web photo presentation system

Currently we convert several existing LAML document styles to the XML-in-LAML framework. As an important language for this material - LENO has already been transformed.

28.5. More information

Lecture 7 - slide 22

There are a number of sources to more information about LAML

- **Academic**
  - The papers available from the LAML Home Page
- **Tutorial**
  - The elucidative LAML Tutorial
- **LAML user/programmer information**
  - The LAML software home page

The LAML home page [laml-home] contains all the papers that have been written above LAML and LENO.

The aim of the LAML tutorial [laml-tutorial] is to introduce various aspects of LAML in a gentle way.

The LAML software home page [laml-software] contains an overview of all the software, and it gives access to a download page where the latest distribution is made available as free software.

LAML can be downloaded as free software from the LAML home page

The exercise below is seen as a typical, practically oriented example where LAML can be used for management of everyday information.

Exercise 7.1. *Bookmark administration*

Exercise motivation: It is not practical to bind web bookmarks to a single machine nor to a single browser. Therefore we will maintain a list of bookmark entries in a file, and generate web bookmark pages from this description.

We will assume that we maintain a *list* of bookmark entries, each of the form

(`bookmark` *URL title category comment*)

The first constituent is a symbol (a tag that distinguishes bookmark entries from other structured data) and the other fields are text strings.

You can find an example of a bookmark list here. See the link in the web version.

The exercise is to complete a frame-based bookmark browser, given a list of bookmarks.You can find an example of a frame-based bookmark browser here. See again the link in the web version. To make it realistic for you to solve this exercises, you are only asked to make the left frame: the category overview frame. The frameset page and the right frame page are already programmed.

You are, however, supposed to understand the pre-programmed frame of the exercise. So start to read through the existing pieces of the program.

You can find the pre-programmed part of the bookmark browser in the zip file bookmarks-zip.zip. Please consult the web version of the material to get access to it. Unzip it, and LAML process the file `bookmark.laml`. Bring up the file `bookmark-frameset.html` in a browser to get started.

You are welcome to extend the solution, for example with a good use of the comment field from the bookmark entry.

28.6. References

[laml-software] The LAML software home page - development version
http://www.cs.auc.dk/~normark/scheme/index.html

[laml-tutorial] The LAML tutorial
http://www.cs.auc.dk/~normark/scheme/tutorial/index.html

[laml-home] The LAML home page
http://www.cs.auc.dk/~normark/laml/

[laml-cgi-tutorial] Guess a Number - A simple CGI program in Scheme with LAML
http://www.cs.auc.dk/~normark/scheme/tutorial/cgi-programming/cgi-programming.html

[kn-calendar] The authors personal LAML calendar
http://www.cs.auc.dk/~normark/cgi-bin/calendar/data/normark.html

[normark02e] Kurt Nørmark, "The LAML Software Home Page", 2003.