`mixer` — maintaining web sites easily

Max Neunhöffer
max.neunhoeffer@math.rwth-aachen.de

15. Juli 2004

Inhaltsverzeichnis

1 Introduction and Basic Idea

If you have a rough idea what the `mixer` is and just want to see quickly how it all works, have a look at the examples in section 2 and then read section 3. The gory details are then covered in sections 4 to 12.
The `mixer` is little Max's version of a „Content Management System". The goal is to help the user to maintain a web site consisting of a hierarchy of web pages in a consistent way. Repeating parts of these pages should each be stored and maintained in a single place and an easily understandable procedure should make the final pages from the input the user enters.

For each page this procedure basically pastes together various files from different places in a well-defined way and replaces certain elements by other stuff. Things like navigation tools for the visitors of the site are generated automatically from available data.
The main design principles are:

- SIMPLICITY,
- well-definedness,
- good documentation,
- the use of standard web technology (XML, XHTML, style sheets).

2 Learning by example

Assume you have a directory `DIR` and the following list of files and directories:

```
DIR/MIXERROOT
DIR/lib/addresses
DIR/lib/config
DIR/lib/default.html
DIR/index.mixer
```

The `MIXERROOT` file is empty, `addresses` is a address database in a certain format and `config` contains some variable settings. Assume `default.html` looks like this:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
  <link href="/lib/default.css" type="text/css" rel="StyleSheet" />
  <style type="text/css" media="screen">@import url(/lib/default.css);</style>
  <title><mixer var="title">Here will be the title</mixer></title>
</head>
<body>
<mixer part="main">Here will be the main part of the page</mixer>
</body>
</html>
```

and `index.mixer` looks like this:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<mixer template="default.tmpl">
<mixertitle>My title</mixertitle>

<h1><mixer part="title"/></h1>

<p>Current semester: <mixer var="semester"/></p>
<p>Author: <mixer person="Max Neunhoeffer" data="name_link"/></p>
<p>Some weird display, possibly navigation:<mixer func="myfunc"/></p>

</mixer>
```

Then the `mixer` does the following for you:
It takes the file `index.mixer`, reads off the name `default.tmpl` from the `template` attribute and starts with `default.tmpl` in the `lib` directory and produces a file `index.html` by replacing things.

The content from the file `index.mixer` (beginning with the `<h1>` is inserted instead of

```
<mixer part="main"/>
```

and the title „My title“ is inserted instead of `<mixer part="title"/>` in the head of the document and in the `<h1>` element.
The value of the variable `semester` is substituted for `<mixer var="semester"/>` and from the address database a link to the personal page of Max Neunhöffer together with his name is inserted instead of

```
<mixer person="Max Neunhoeffer" data="name_link"/>
```

It is even possible to put the output of a user defined function `myfunc` in the final page instead of

```
<mixer func="myfunc"/>
```

This makes it for example possible to automatically generate navigation tools for a whole web site.
This document is the manual to the `mixer`, a program that achieves.

3 If you are in a hurry

Here is a table of replacements, that are performed:

`<mixer part="main"/>`	→	the content of the current `.mixer` file.
`<mixer part="title"/>`	→	the title of the current page from the `mixertitle` element.
`<mixer var="uvw"/>`	→	the value of the variable `uvw` in the configuration database.
`<mixer part="abc"/>`	→	the content of the part `abc` of the current page which is found in the file `xyz.abc` in the same directory, if we are working on the file `xyz.mixer` currently. If the file `xyz.abc` does not exist, a warning is given out. This warning can be switched off by specifying an additional attribute `needed="no"` in the `mixer` tag.
`<mixer person="ID" data="KEY"/>`	→	the entry under the key `KEY` of the person with id `ID` in the address database
`<mixer func="hij"/>`	→	the result of the user defined function `hij`
`<mixer klm="xyz"/>`	→	the result of the user defined function `klm`. Here `klm` must not be equal to one of the other predefined attributes.
`href`-attributes	→	links not containing a colon and starting with a slash are interpreted as links relative to the root of the site and are replaced by links relative to the current file, variable names in links, which are enclosed in double curly brackets, are replaced
`<link href="opq.css"/>`	→	the value of the `href` is replaced by the value of the (optional) `oldstyle` attribute of the `mixer` element in the `.mixer` file.
`<style>@import url(opq.css);</style>`	→	the link is replaced by the value of the (optional) `style` attribute of the `mixer` element in the `.mixer` file.

4 Notation

The `mixer` feels responsible for a full subtree of the file system. This tree is called „the web site“ and the absolute path name to its top is called `MIXERROOT`. Individual HTML-files within this subtree are called „web pages“.

5 Installation of the `mixer`

First note that the `mixer` needs Python version 2.2 or newer and that this must be the version of Python coming up if one calls `python`.

Untar the archive `mixer.tar.gz` (possibly your version contains some version number). This will create a directory `mixer` where the code resides. The `mixer` is written in Python, however it uses an extension of the Python interpreter which is written in C and uses the C programm `rxp` as an XML parser. Therefore some compilation is needed. This is done by doing `make` in the `mixer` directory.
The only further step for installation is to create a symbolic link from any position which is in your `PATH` (for example `/usr/local/bin`) to the file `mixer.py` in the `mixer` directory.

6 First steps for a new web site

To get things going, you need at least:
A directory with an empty file with the name `MIXERROOT` and a subdirectory `lib`.
You need a file with the name `addresses` in the `lib` directory. It is a valid XML file with only one element of type `addresses` that contains elements of type `person` with arbitrary attributes.
You further need a file with the name `config` in the `lib` directory. It must be a valid Python module. You can define variables in there in Python notation. All variable values must be strings.
The file `funcs.py` in the `lib` directory is optional and only used if you want to use user defined functions.
To see the `mixer` doing something you need at least one template file and a file with `.mixer` at the end.

7 Finding the `MIXERROOT`

The `mixer` can be invoked from anywhere within the file system tree of the web site. It determines the `MIXERROOT` by looking upwards for an empty file with the name `MIXERROOT`. Here are the details:
After invocation `mixer` first determines an absolute path to the current working directory. This is done by looking at the environment variable `PWD`, which is set by most shells to an absolute path to the current working directory. However, this can contain symbolic links embedded in the path. So `mixer` checks whether this directory is the same as the directory that the C-library function `getcwd()` returns. If both paths refer to the same physical directory, then the value of `PWD` is taken, otherwise the result of `getcwd()`.
The `mixer` then goes up in the file tree by removing parts at the end of the path, until it finds an empty file with the name `MIXERROOT`. The path pointing to the corresponding directory is the `MIXERROOT`.
The reason for taking the value of `PWD` is that symbolic links within this path should be preserved. The comparison with the result of `getcwd()` is done just to be save. Note that the file system subtree below `MIXERROOT` **must not** contain symbolic links to subdirectories.

8 How does the `mixer` put together web pages?

The `mixer` first reads the configuration database in `MIXERROOT/lib/config` and then the address database, which is in the file `MIXERROOT/lib/addresses`.
After that it interprets the file `MIXERROOT/lib/funcs.py` (if it exists) as a Python script to get hold of the user specified functions. Then it walks recursively through the whole subtree below `MIXERROOT` and does the following job on all files ending in `.mixer`:
Assume it works on a file `xyz.mixer`, which has to be a valid XML-file with top level element of type `mixer`, having an attribute `template`. The value of this attribute has to be one of the template documents (without any path). The referenced template document also has to be a valid XML-file and has therefore a natural tree structure. The `mixer` starts with this tree and recursively walks through it, replacing the subtree corresponding to the element `<mixer part="main"/>` by the tree defined by `xyz.mixer`. Directly after the replacement this tree is walked through and only after that is done the walk through the first tree continues.
During these walks various replacements of subtrees are done, which are explained in the following sections. As described in section 10.2 for „part substitution“ other files with names like `xyz.ABC` are used, where `ABC` is replaced by the name of the requested part.
There are possibilities to insert the value of a variable defined in the configuration database, to insert values from the address database, to insert other parts or to insert the result of a user defined function.

After all these replacements, the resulting tree is written out as a valid XHTML document (provided the input was correct) to the file `xyz.html`.
The `mixer` usually only works on those files `xyz.mixer` which have a newer modification date than the corresponding file `xyz.html`, unless the command line option `-f` for „force“ is used. In the latter case or if any of the template documents is modified more recently, the complete site is rebuilt.

9 Template documents

Template documents play the role of a common frame for many web pages. They are typically ending in `.tmpl` and can reside anywhere in the file tree below the `MIXERROOT`. Their format is easily described: A template document must be a valid XHTML document, apart from certain additional elements of type `mixer`, which are replaced during the work of the `mixer` as described below. However, to make sense, a template document should adhere to the following conventions or rules:
A template document should **not** contain a `DOCTYPE` declaration. This is inserted by the `mixer` before writing out the HTML-files. The reason for this is that with the `mixer` elements it **is not** a valid XHTML document and with the `DOCTYPE` declaration the XML parser within the `mixer` would complain.
A template document must have an element of type `title`, however its content does not matter. This is a rule from the XHTML standard.
A template document should contain an old style and a new style declaration of a style sheet, as follows (in this order!):

```
<link href="/lib/default.css" type="text/css" rel="StyleSheet" />
<style type="text/css" media="screen">@import url(/lib/default.css);</style>
```

Note that the seemingly absolute links will be replaced by the `mixer` as described in section 10.6. Modern browsers will read both declarations and the values in the second will take precedence over those in the first. Netscape version 4.xx (and probably earlier) will not read the second. This is a way to deal with the inherently broken implementation of cascading style sheets in netscape version 4.xx. If need be, one can easily specify a different stylesheet for netscape 4.xx.
A template document should somewhere in the body contain an element `<mixer part="main"/>`. Otherwise all pages referring to this template document will be equal and the main part is not included at all.
The template document used for a certain `.mixer` file is determined by the `template` attribute of the top level `mixer` element in the `.mixer` file. The attribute value should not contain a path but only a filename. This file is searched from the position of the `.mixer` file upwards until the `MIXERROOT` and the first file found with this name is taken. If no file is found the `mixer` also searches in `MIXERROOT/lib`.

10 What replacements does the `mixer` perform?

During its walk „through the tree“ (see section 8), the `mixer` replaces all elements of type `mixer`. Actually it replaces the whole subtree below such an element. Therefore with respect to the final result it does not matter, whether you write

```
<mixer part="main"/>
```

or

```
<mixer part="main">Here will be the main part.</mixer>
```

However since most browsers will ignore the `mixer` tags that they do not know and display only the text „Here will be the main part.“, the second variant has an advantage: You can look at the template document with a web browser and will see the place, where later the actual content will be placed. This hint of course holds for all replacements described in the following sections.

10.1 Variable substitution

Any element of type `mixer` having an attribute `var` with value `x` is replaced by the value of the variable `x` in the configuration database.
Example:

```
<mixer var="a"/>
```

The configuration database is just the file `MIXERROOT/lib/config`, which must be a Python script that only sets some variables to string values. Note that variable assignment is done with `=` in Python and that string values have to be enclosed in either single or double quotes. If you want to define string constants that contain more than one line, you have to use either triple quotes. Some examples:

```
a = 'Max'
b = "Till"
c = '''Hi
there!'''
d = """This even contains single quotes ' and
multiple lines!"""
```

There is a special variable with name `today` which is replaced by the result of the C-library function `ctime`. Note that of course this will give you the date of the time when the `mixer` ran last!
Another special variable with name `timestamp` is replaced with the output of `ctime`, called with the modification time of the `.mixer` file of the main part.

10.2 Part substitution

Assume the mixer currently works on a file `x.mixer` in some directory `dir`.
Any element of type `mixer` having an attribute `part` is replaced by a full tree from another file. If the value of the `part` attribute is `main`, the tree from `x.mixer` is used. If the value is `title`, a special case occurs, which is described in the following section. For all other values `y`, the content of the file `x.y` in the directory `dir` is taken.
If the file `x.y` does not exist, a warning is given out. This warning can be switched off by specifying an additional attribute `needed="no"` in the `mixer` tag.
Example:

```
<mixer part="main"/>
```

Note that in all cases the inserted subtree is traversed recursively before the rest of the first tree.

10.3 Parsed variable substitution

Any element of type `mixer` having an attribute `parsevar` is replaced by a full tree from the variable that has the name of the value of the attribute. That means that the (string) content of the variable is parsed (an XML file header is prepended indicating ISO-8859-1 encoding and one pair of tags `<mixer>` and `</mixer>` is put around the content of the variable). The resulting tree is then written out recursively, following the standard rules.
Note that this of course can lead to infinite recursion, as variable substitution, part substitution, parsed variable substitution, and all other substitutions are done recursively.

10.4 The title of a page

There is a special case of part substitution, which was already mentioned in the last section, namely if the `part` attribute has the value `title`. Very often the title of a page will occur not only in the `title` element of the head of the page, but also in some heading in the main page. Therefore the following mechanism was invented.
When the main part is read from a `.mixer` file, the top level of the corresponding tree is scanned for an element of type `title`. If it is found, it is removed in this tree and stored separately. Its content is then placed not only into the `title` element in the head of the page but also inserted at all places, where an element like `<mixer part="title"/>` appears. Note that recursive replacement takes place within this title subtree, such that for example variables can be used inside.

10.5 Style sheets

To configure the usage of different stylesheets conveniently some extra magic has been implemented in the `mixer`. If the top level element of a `.mixer` file has an attribute `style`, its value is placed into the new style stylesheet declaration of type `style` in the head of the document. Likewise, the value of an attribute `oldstyle` is placed into the old style stylesheet declaration of type `link` (see section 9).

10.6 Links

For all links in the trees a convenience procedure is performed. All values of `href` attributes are considered to be links. In addition the value in the `@import url(...);` statement in the `style` element in the head is also considered a link.
First a special form of variable substitution is performed: If the link contains constructs in double braces, the text between the braces is taken as the name of a variable from the configuration database and the brace expression is substituted by the value of the variable.
Afterwards the link is changed into a relative link in the following way:
If a link contains a colon, it remains untouched, because it is considered to be an external link, probably containing `http://` or a similar thing.
If a link does not start with a slash, it remains untouched, because it is considered to be an internal, relative link „as is“.
If a link does not contain a colon and starts with a slash, it is considered to be an internal link, which should be relative to the current position in the end. It is interpreted as a path relative to the `MIXERROOT` and is changed to a relative link from the current position, usually by prepending a certain number of repetitions of „`../`“ for „go one up“.
This last feature is convenient, because one can always refer to other documents via their „absolute“ path with respect to the `MIXERROOT` and gets automatically relative links in the end.

10.7 User defined functions

If the `mixer` finds a construct like `<mixer func="xyz"/>`, it calls the function `xyz` which must be defined in the file `MIXERROOT/lib/funcs.py` which is interpreted at startup time. This function gets five arguments. The first is an absolute path name to the `MIXERROOT`, which ends in a slash. The second is a path name leading from the `MIXERROOT` to the directory in which the current `.mixer` file resides. The third argument is the name of the current `.mixer` document without the extension `.mixer`. The fourth argument is the complete tree of the main part in the memory representation described below. The fifth argument is the current subtree in the same form.
The function must return something which can be worked on by the „walking“ routines in the `mixer`.
The return value can be a string. In this case it is just put at the place of the `mixer` element. If it is a list, all entries in the list are inserted instead of the `mixer` element (and therefore are again subject to the same requirements).
The return value also can be a full subtree in the memory representation of XML-trees within the `mixer`. This representation works as follows: An element is represented by an object in the class `xmltree` which is defined in the module `maxml`. This class has four data entries: `type` is a string containing the name of the element. `attr` is either `None` or a Python dictionary containing the attributes of the object. `subs` is either `None` for an empty element or a list of children. Those children can either be strings or subelements in the form of objects in the class `xmltree` recursively.
Note the difference between the value `None` and an empty list: The first indicates that the element had only one opening and closing tag like `<element/>`, the second indicates, that it has opening and closing tags, but no children.
The last entry is called `meta` and contains meta data like source positions and is not used for further processing.
Note that `mixer` elements with function calls need not be empty. User defined functions have access to the subtree via the fifth argument.

10.8 Short form of call to user defined functions

As convenience for web site authors there is a short form to call user defined functions:

```
<mixer myfunc="Argg!"/>
```

If in an `mixer` element has exactly one attribute key and this is not among the predefined names described in this section, the `mixer` assumes that this attribute key (in the example above „`myfunc`“) is the name of a user defined function, which is called as in the long form described in the last subsection.

11 The address database and its applications

The address database in the file `MIXERROOT/lib/addresses` is of course again an XML-document. It must consist of a single element of type `addresses`, which contains a sequence of elements of type `person`. Those latter elements all have to have an `id` attribute for identification and may have any other attributes. Standard attributes in use are:
`name, formalname, title, department, university, building, street, city, count, zipcode, country, www, email, fax, phone`
If one uses the attribute `sameaddressas` with the `id` of another person as value, the address information of the other person is copied unless specified otherwise.
The information in the address database can be inserted into web pages by using elements like

```
<mixer person="xyz" data="abc"/>
```

where `xyz` must be the `id` of a person in the database and `abc` must be an attribute specified for that person. Note that some additional attributes are generated automatically, if the necessary data is available:

`name_link`	for a link to the person’s web page which has his name as text of the link.
`name_link_email`	for the above link together with a clickable email address.
`email_link`	for a „mailto:“ link with the email as visible text.
`title_name`	for the full name with title.
`address`	for the full address consisting of the following entries, if given: `department`, `university`, `building`, `street`, `county`, `city`, `zipcode`, `country`.
`contact`	the address as in `address` plus an email address, if given.
`name_city`	the name and the city, if both are given, otherwise only the name.
`name_link_city`	the name linked to the person’s web page, and the city, if both are given, otherwise only the name.

12 Automatic generation of site navigation

The current distribution of the `mixer` contains a sample implementation of navigation generating functions in `MIXERROOT/lib/funcs.py`. In this approach one assumes that the whole site has a „spanning tree“, which is mirrored in the file system tree. Every node in this tree corresponds to web page. Every internal node (i.e. not a leaf) corresponds to a directory in the file system and every leaf corresponds to a page within the directory of the parent node.
This tree is configured by putting a file with the name `tree` in each directory within the web site. Such a `tree` file must be a valid XML-file with one top element of type `node`, that has an attribute `file` which contains the name of the web page corresponding to the node of the tree belonging to this directory. The top element contains a sequence of elements of type `entry` which declare the children in the tree. Note that the order of the entries matters for navigation purposes. Each `entry` node has an attribute `file`, the value of which refers to either a subdirectory (if the child is not a leaf) or to a file in the current directory (if the child is a leaf). The content of the `entry` element is the text that should appear in the navigation tools.
These are the two functions which generate such navigation tools:

`<mixer func="maketop"/>`	for a function producing a top line with links to the children of the root node
`<mixer func="makeleft"/>`	for a function producing a vertical bar visualizing the current position in the tree by displaying the current path and the nodes in the vicinity. The idea is that of a bar appearing to the left of the main content by using a table.