♢ ♢ ♢

Extending ExTEX

Simon Pepping

abstract. What can be done after the completion of ExTEX? I describe a dream, some results, and some further ideas.

keywords: ExTEX, DSSSL, file location, extension of ExTEX, primitives in ExTEX

SGML, DSSSL and TEX

Complex architectures have always appealed to me. It is no wonder then that the system SGML – DSSSL – typographical backend found in me a believer. Sebastian Rahtz's `jadetex` is a good implementation of TEX as the typographical backend. But ever since I had a look at it and at the way the `jade` DSSSL engine communicates with the backend, I have had this dream of a direct communication between the DSSSL engine and TEX, and of a direct implementation of flow objects in TEX.

One might think that this dream has been made obsolete by modern developments: SGML is out, XML is the new king. But I do not think things have really changed in the style specification area: The style language is now called XSL, its abstract page specification objects are now called formatting objects, but the idea has remained the same.

Obviously, I was not sufficiently skilled as a programmer to realize such a program. I also had the idea that with the current TEX program it would be impossible, and anyway I had little desire to work through the complicated canonical route to hack the TEX sources.

Therefore I looked forward to the release of the ongoing ExTEX project, so that I could try to realize my idea in a system with a more extensible structure. When I got the program in March, I started to try and understand its structure, and see where I should start to plug in my changes. That gave rise to some interesting conclusions.

TEX input without macros

When the `jade` program talks directly to the typographical backend, it makes calls to subroutines that start and end the flow objects, subroutines that register new values of characteristics, and subroutines that receive textual data. Those calls specify the layout in terms of abstract flow objects. From there the typographical backend should take it, and produce pages according to the abstract specification.

TEX, on the other hand, requires input in terms of its macro language. That has

been a great benefit. While the TEX program itself has proved hard to extend, the macro language has provided programmers/users with ample opportunity to write extensions.

The interprogram communication sketched above bypasses the macro language completely. As a consequence none of the extensions created in the 80s and 90s are available. When one looks at `jadetex`, one sees that it makes a good effort to pull in all major packages written for LATEX2ε. So, when we are not able to use those, we have a big problem. It is even worse: we have even bypassed plain TEX; indeed, we do not even have plain TEX's output routine, because that is specified through control sequences as well.

Karel Skoupy, the author of ExTEX, tells me that we do not even have a typesetting engine; we have no more than a library. It is the macro language that glues it all together into a typesetting engine. So my idea comes down to doing away with that glue!

That is where my plans have stalled. I will need some bright ideas to find a way forward.

Serving the files to ExTEX

When I had the ExTEX program, and took part in some discussions between its author and its users, it soon became clear that there was a problem in the way ExTEX finds the required files in the TEX distribution. Java, being a platform-independent language, has problems communicating with the environment. The chosen solution was to launch `kpsewhich` as a separate process to find the files for ExTEX.

While the `kpathsea` library has become a *de facto* standard, I am not ready to accept it as the only way to locate files in a TEX distribution, now and in the future. And therefore I do not think it a good idea to hard-code the use of `kpsewhich` into ExTEX. I prefer a separation between the typesetting engine and the TEX distribution, even though TEX has built-in file locating capabilities. I want a file locator architecture that is configurable by the distribution. The setup of Java's security mechanism, which is extremely configurable, with the possibility to plug in third party implementations of one or more functions, showed me how this could be achieved.

I constructed what I call the pluggable file locator architecture. ExTEX creates a `File Locator` object which defines the required file location functionality and its API. One or more implementations of that functionality may then be written, and added to ExTEX as modules. Start-up options determine which of the available implementations is actually used.

Basically, it works as follows. On the command line one has to tell ExTEX which file locator implementation one wishes to use. This can be done using the java property `nts.filelocatorclass`. Or it can be done in a configuration file, whose name should be communicated to the application using the java property `nts.filelocatorconfig`. An added bonus of a configuration file is that its path is communicated to the implementation, a feature which I use below in the `kpathsea` implementation. Extra arguments can be passed to the implementation. For more information, see the doc-

umentation in the java code itself (which is extractable with the javadoc tool). The idea is that the TEX distribution configures the command line to its needs, so that this is transparent to users.

Here is an example of the startup line of ExTEX using a file locator configuration file:

```
java -Dnts.fmt=latex \
 -Dnts.filelocatorconfig=/usr/share/TeX/bin/ntsfilelocator.cfg \
 Nts latex-file
```

The configuration file contains the following line:

```
nts.filelocator=<package>.<file locator implementation class>
```

Here is an example of a startup line directly providing the file locator implementation class. This class has a single constructor argument:

```
java -Dnts.filelocatorclass='<package>.kpsewhich \
                             /usr/share/TeX/bin/kpsewhich' \
 Nts tex-file
```

kpathsea

Of course, the *de facto* standard `kpathsea` was my first implementation. Since it is written in C, I used the Java Native Interface (JNI) to access it from the Java code. Hacking `kpathsea` turned out to be relatively easy. I took the code of `kpsewhich`, removed all code that is related to the command line options, added the functions that implement the Java interface, and I was almost done.

There was one complication: `kpathsea` is very much written for the situation where it is linked into an executable. Here the executable is `java`, which is of little help. So I have to supply artificially the name of another program, one in the `kpathsea` bin directory. I use `kpsewhich` for this purpose.

To my surprise, modifying in the `kpathsea` code the name of the executable from `java` to that of a `kpathsea` program:

```
kpse_set_program_name(NTSprogpath, NTSprogname);
```

at first did not work. It turns out that on `glibc` systems `kpathsea` does not use the two arguments of this function. The `glibc` library catches the program names and those are used by `kpathsea`. Therefore I have to reset the `glibc` program names:

```
program_invocation_name = NTSprogpath;
program_invocation_short_name = NTSprogname;
```

The file locator configuration file itself can be put in the `kpathsea` bin directory and used as a pseudo `kpathsea` program. When the first of the above ExTEX startup lines is used, the `File Locator` class communicates the path of the configuration file to the constructor of the implementation, and the latter uses it as the program path. This feature really embeds the file locator interface to ExTEX in the `kpathsea` setup of bootstrapping the distribution from the path of the executable.

TEX file server
As became apparent during discussions, Java is not good at communicating with the environment of the computer on which it runs. On the other hand, it is excellent in communicating with the networked world. So the idea presented itself to serve files over a network.[1] This was my second file locator implementation.

It was my goal to write a simple proof-of-concept. So I wrote a simple TEX file server using Java's ServerSocket class, and a simple TeXFSClient class as the file locator implementation that uses the TEX file server to get its files. The protocol is also simple:

Open TeX session The client sends a handshake to the server, to make sure the server can be found and is alive.

Open TeX file The client requests a file. It communicates the characteristics which are familiar from `kpathsea`: file name, format, must exist and program name.

This implementation basically demonstrates the idea. But it has some serious limitations:

- The session is not persistent, each file is requested in a new connection.
- In `kpathsea` it is not possible to change the name of the format, so once opened, the TEX file server serves files for only one format.

If this were to be turned into a serious tool, there is still a lot to be done. There are several possibilities to set up a more robust TEX file server. A good way to do it is over HTTP. It makes use of an established protocol, which is implemented in a number of excellent web server systems. To communicate TEX specific messages, one could use a CGI- and XML-based protocol, such as that recently developed by the Open Archives Initiative (see `http://www.openarchives.org/OAI/openarchivesprotocol.htm`).

Other extensions to ExTEX

LATEX2ε has some outstanding features which deserve to be separated from LATEX's document style features and made available to *all* TEX users:

NFSS Ever since the early 90s LATEX's font selection scheme has made it relatively easy to manage an ever growing font collection. Adding a new family of fonts is done in a systematic and transparent way.

graphicx/s providing the much needed integration of LATEX with graphics.

babel making (La)TEX multilingual.

LATEX has implemented a number of extended primitives, such as `\newcommand`, or the control sequences of the `ifthen` package, which make macro programming more robust and easier. These could be implemented as true primitives within ExTEX.

LATEX provides an interface to document classes, and for that purpose defines many LATEX primitives, such as `\@startsection`, the `counter` commands, the `option` commands. That interface can be realized in ExTEX itself, as a module. LATEX users can

[1] I understand that the file server idea has also been put forward within the NTS steering group.

use it, others can leave it alone.

These facilities (and LATEX itself) have been realized using TEX's extension mechanism, its macro language. This is an astonishing feat, in view of the fact that the macro language is hardly adequate as a programming language. Now that we have an extensible program, it seems a good idea to program these facilities as modules. As a programmer, I believe that the use of a programming language would make it much easier to achieve the desired logic.

Epilog

ExTEX has been awaited for a long time, and its realization is not entirely satisfactory: it is slow and resource hungry. But despite these drawbacks, we should appreciate that it is the first complete reimplementation of TEX ever made.

Its release provides us with a tool that we can play with, modify, and extend, more readily than we can do with TEX. At the very least, it is a sandbox on which new ideas can be tested. I hope the recount of my efforts provides inspiration to others to start working with the system.