WP2LaTeX programmers manual

1 Purpose of this text

This document is intended for programmers. You will find here only programming notes.

I would like to reveal here several things from internal structure of WP2LaTeX. They might be useful for following purposes:

1. You want to add new file format support for conversion into the \LaTeX
2. You want to convert WordPerfect fornat into format other than \TeX
3. You want to study my code for better understanding how to program effectivelly

Discussion

1 - this feature is currently supported quite well. You could write your new module that will call functions from pass1.cc. You could be inspirated by one of already existing conversion modules.

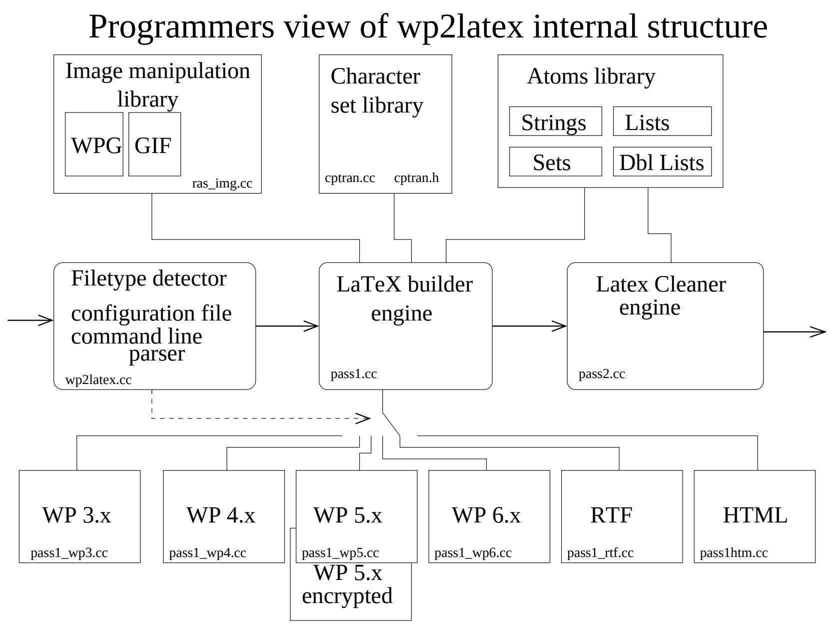
2 - this way might be more complicated. I will briefly explain why. There are a lot of very simple conversion programs. They convert from one typesetting language to the other one simply by replacing symbols from one language with their counterparts froum second language. Unfortunatelly these programs do not obtain good conversion quality.

WP2LaTeX do not provide such simple conversion and it behaves like state optimalisation automate with optimization step. Most od documets is abble to compile with LaTeX immediatelly after their conversion from WordPerfect. But on the other site the switching of another output format might be quite hard.

If your program could import LaTeX it might be more easier to convert through intermediate LaTeX format with some options turned on.

3 - yes, you could study my sources and this documentation. Sorry for small amount of comments.

2 WP2LaTeX internals



At a first glance you could see that wp2latex is a framework with optional conversion modules. Each conversion module corresponds to one supported fileformat. This was necessary during design, because Word Perfect file formats are totally different. Because of such a general design, there is very simple to add another totally different file format like RTF or HTML.

2.1 File format detection

Because all supported formats differs strongly, WP2LaTeX is using sophisticated file format detector. The file format detector could detect appropriate conversion module in most cases. Please note that some file formats like WP4.x cannot be surely detected and it is possible only heuristic approach.

File format detector returns so called format descriptor. It is a short human readable string - see FFormatTranslator object declaration. The table of available conversion modules is searched for this string. If it is found, the corresponding conversion module is used. Otherwise the error message is printed.

The file format detector does not depend on another parts of WP2LaTeX code and could be used inside another programs as well.

2.2 Conversion modules

From the programmers view every module must have one static object type *FFormatTranslator*. Its constructor is automatically called, so you do not need to modify source code during adding or removing another conversin modules. How many modules are passed to the linker so many modules will be used inside executable.

Real example:

```
/*Register translators here*/
int Convert_pass1_602(FILE *FileIn,FILE  *table, FILE *StripOut,FILE
*LogFile,FILE *ErrorFile,void *Wrapper);
FFormatTranslator Format602("T602",Convert_pass1_602);
```

Conversion modules are using common libraries strings, lists, character library for conversion character sets and image library for handling images. Most important is the “LaTeX primitive library” that is stored inside file pass1.cc.

2.3 Latex cleaner

When a convertor stupidly translates a sequence of tokens (tags, objects) from one text processor to another, the results are usually not good. For example crossing bold face and center enviroment could produce unexpected and unwanted results.

LaTeX optimizer solves so called paragraph property changes. It knows character properties at the beginning and at the end of every line (stored inside .TAB file). When the paragraph property is changed, all character attributes are closed. Then the code sequence for changing enviroment is emitted and finally the proper character properties are reopened.

The second task of optimizer is removal useless sequence of control codes like: {\bf {\it } }. Several Word processor like WP contains these sequence inside documents. They are not visible inside WYSIWIG mode, but they are pretty clumsy inside LaTeX.

2.4 Atoms library

This library contains a set of primitives for working with strings and lists. There are a lot of libraries that are performing these tasks, but unfortunatelly I had problems with their portability (Borland C++ AnsiString, MSVC CString) to another platforms or compilers, availability of source code, licence cost, code size etc. Thats why I have used my own tinny library. Because the library has its own documentation, I will redirect reader to it.

2.5 Image manipulation library

You could note that there are a lot of libraries for image conversion available. But unfortunatelly no of them support WPG (Word Perfect native format). That is why I used my own library (its previous name was pictures). Because the library has its own documentation, I will redirect reader to it.

2.6 Character library

There are huge amount of different encoding 8bit, 16bit and the wp2latex should understand them and generate appropriate LaTeX sequence during conversion. Wordperfect itself has 3 different encodings, that are not supported by any publically available library. The conversion among engodings was pretty necessary.

The character library is much more generator of source code than the true library. It reads character encoding table i.e. wp6.enc and emits source code that contains conversion table. Such conversion table is space optimised and it runs very fast. Conversion tables are generated during *make* phase of WP2LaTeX package.

The character.cc is the LaTeX glyph generator that is build on the top of the character library.

3 Supported compilers

The WP2LaTeX was designed to be portable. But unfortunatelly I do not have all compilers to test its portability. Even if the C++ is normalised, there are still small nuances that lead into breaking of functionality or inability to compile.

The most painful difference was existence of different endians and different size of types like int, short int, long int. There are defined types BYTE, WORD and DWORD inside file types.h. Please consider to review this file when you aim to recompile WP2LaTex on the different platform. Anyway, WP2LaTeX has builtin its ow check and will not run when it is compiled with bad BYTE, WORD or DWORD sizes.

Currently supported compilers are:

- MS Visual C++ 6 (there also exist free demo version)
- Borland C++ 3.1 (only real overlayed target)
- Borland C++ 5.5 (freely available)
- GNU C 2.95 (Tested DJGPP and Linux targets, reported that Solaris and AIX should also work)
- GNU C 3.0 (Tested DJGPP and Linux targets)

Please let me know when you succeed to compile WP2LaTeX with another compiler.

4 Contents

Contents