Appendix 8 - Future Versions

(Alternate heading - Roy's "Wish List")

This section indicates what is planned to go into future versions of ReWrite. These plans are subject to change (feedback helps), and there is no timetable of when future versions are released, as it depends on my workload and my level of interest (again feedback may help the latter).

Note that well written (not doing anything silly with the types) source code will have nearly compatible source code with future versions, but that object code will not be compatible - recompiling will be necessary.

ReWrite 0.3

• More unstructured types

- short integers (16 bit)
- reals

• A module/project structure

In the current version, there is a big list of function names that can't be used because they are used by the compiler. I hope to change the current model of one project, all linked together, to another model where each module specifies what modules it needs, and the compiler is just another module that doesn't step on anything else.

At the moment there is a primitive module structure, but this will not be retained, so is undocumented.

Included in this update will be ways of specifying in which order rules are checked.

• Macro's

The current project supports macro's, but they are clumsy, and so undocumented.

• New higher level functions

`Map`, `Apply`, `Fold`, `Select` etc.

• Access to intermediate code

Again, the current project supports access to the intermediate code for low level functions and very fast programming, but this is currently full of holes, and the language is not even well defined, so is again undocumented.

This will require:

- a well defined intermediate language;
- a better register allocator (the current one cannot handle backward branches).

• A smaller, much faster compiler

This should result from some of the other improvements being discussed.

• More functions

For example, file handling and other I/O.

• Some internal improvements

- function call optimisation

Functions that are internal to a module need not use the full calling conventions; for example parameters could be passed in registers to hidden functions.

- a better register allocator

Dealing with backward branches is required.

- a better memory manager

At the moment, cells greater than 40 bytes aren't allowed - I have the code to deal with this problem, it's just a matter of converting it from assembly to intermediate form.

- better machine code

This includes some low-level optimisation.

ReWrite 0.4+

• More structured types

- arrays
- strings
- records
- graphs - this will really make optimisation code easier.
- arbitrary sized integers

There is much efficiency to be gained by using specialised structures rather than lists.

• FPU Support (maybe)

(If I ever get an FPU - on the other hand, look at all those applications that broke when the PowerMac came out)

• A full Input/Output library

- Use of a GUI interface from inside ReWrite.

• Some sort of object-like structure

This will not be the standard objects as found is Object Oriented Pascal - it will be something having many of the same properties, but designed for this environment.

• A new user interface

This would be written in ReWrite, so the whole application is written entirely in ReWrite. This will avoid some of the current limitations imposed by Pascal and TextEdit.

• A debugging environment

• Port it to other machines (particularly a PowerMac)

At the moment, ReWrite is a 'spare time' project, so it will only be developed on machines that I own (that is why a Classic was used until recently), so while it is designed to be easy to port, it will probably not happen until I upgrade my machine again, and since I have just done that, this will be a while away.

• Internal improvements

- everything fully relocatable

This allows easy memory compaction, and since everything would need to be able to be tracked down, multitasking on the same memory space becomes easy.

- control/data flow optimization

Big improvements to code speed can be made here.