Project Roadmap

Table of contents

1 Project Roadmap.....2
2 Release Dates.....2
3 Features.....2

1. Project Roadmap

This file describes the intended release schedule for the project. It should be noted that as this project is a 'hobby' type project, fitting itself in around other, more pressing work, any timescales should be taken with a pinch of salt.

2. Release Dates

Preliminary release guesses.

Version	Release Date	Notes
0.1	May 2008	Initial alpha release This release is aimed at getting basic functionality correct. Efficiency is not going to be a concern. The hope is that it will be fully usable, and will be an opportunity to validate the APIs are sufficient. Also, I hope to see if anyone is interested in this system. There will likely be some pain involved in installing the system. There might be some issues running it on different operating systems. It might not be thoroughly tested.
1.0	January 2009	First release All the things should be done before I'm happy for people to use the system in anger.
2.0	January 2010	Second release Really this is a repository for a wish list of things I don't expect to get round to in a hurry. Not all of them are difficult, just 'nice-to-have's rather than 'must-have's.

3. Features

Features slated for particular releases (should really use a proper feature/bug base)

Feature	Priority	Status	Notes
Version 0.1			
Version 1.0			
Error handling	High	Not started	I come from a Java and C++ background. I still don't know how to do error handling well in C. Really, this task just means finding someone who's been doing C in big projects for a bit longer than me and getting their input.
Test cases	High	Not started	Says it all really.
Documentation	High	Not started	Need tutorials and API docs and so on.
ltdl	Medium	Not started	I need to move from using libdl to libltdl. That should allow this system to run on more systems that just Linux.
Makefile	Medium	Not started	My make files are crap. I do actually quite hate make. They need to be beefed up and have to manage C dependencies. It's quite possible that the feature below will take care of this.
GCC style release process - autotools: automake, autoconf	High	Not started	Have to make install as easy as possible. Plus, I have hard coded dir names in my makefile! They have to go.

Version 2.0			

Scripting	High	Not started	This will embed scripting into the system. It will support at least ECMAScript and possibly more languages.
			
			You will be able to have scripts execute in the different phases of the plugin lifecycle. Possibly, each plugin would get it's own, unshared scripting engine instance.
			
			`<script language="javascript"> <setup> var x = 10; print( "Hello plugin folk" ); function foo() { print( "Foo: x =", x ); } </setup> <start src="script.js"/> <shutdown> x = 20; foo(); </shutdown> </script>`
			
			You will be able to write extension points in a script. Essentially, this means you can have:
			
			`<extension-point id="com.acme.my-extension" language="javascript"> function extend( specification ) { // specification is the XML of the extender`

			`}`
			`</extension-point>`
			
			Working out what to do for extensions might take a little more thought.
			
			`<extension point="com.acme.my-event" language="javascript">`
			`function callback() {`
			`// Hmm, but what about the types?`
			`}`
			`</extension-point>`
			
			There needs to be some clever way for dlls to be registered for scripting classes etc. This will need some thought.
			
			There will also need to be a way to know when objects have died in non garbage collected code. This might be tricky for GCC.

Cacheing	High	Not started	At the moment the plugin files are all loaded whether they are lazy or not. It should be possible to cache information about the plugins and only reparse files that are new. There may be some difficulties because plugins might have variables expanded from the environment which could cause dependencies that can only be known at runtime. Possibly this might be fixed by explicity marking plugins

			with an attribute `cacheable="true"`.
Stylesheets	Low	Not started	We could allow `<?xml-stylesheet?>` directives. These might enable people to write large, complex plugin files more easily. I'm not sure if this is necessary or even desirable.
Backwards compatability support	Medium	Not started	At the moment we can only have one accepted plugin document format. It should be possible to have multiple versions co-existing. This would mean that all the PluginManager's extensibility stuff would be packaged up into objects. You would then build plugin parsers with different element handlers. They might also be each derived from different base parsers so that the plugin system itself can evolve. This is all very well, but I think quite a few people would have to be using it for this to be worthwhile.