Using CMake With FireBreath

- Introduction
- How cmake works in FireBreath projects
    - Design Goal
    - Running cmake
        - Location of project files
        - Location of user-created plugins
        - Building example projects
- Making changes to your project
- Problems solved using cmake

Introduction

One of the more common complaints that we receive about the FireBreath project relates to our use of cmake to generate the project. This page is intended to help explain the reasons for using cmake, why we chose it, and how much worse this would actually be if we **didn't** use it.

How cmake works in FireBreath projects

Design Goal

FireBreath has a primary design goal to wrap as much of the browser-specific logic of plugin development as possible, while not impeding the ability of the plugin developer (that's you) to create whatever they want. In order to do this, it is important that the plugin developer should never have to modify any project but their own.

Running cmake

When creating a plugin, you never run cmake directly. Instead, there are a set of "prep" commands in the root of FireBreath that allow you to generate the project files. As of the time of this writing the prep commands (and the tool they generate projects for) are as follows:

- `prep2005.cmd` - Visual Studio 2005 (windows only)
- `prep2008.cmd` - Visual Studio 2008 (windows only)
- `*prep2008x64.cmd` - Visual Studio 2008 (windows 64-bit only, 64-bit plugin build)
- `prep2010.cmd` - Visual Studio 2010 (windows only)
- `*prep2010x64.cmd` - Visual Studio 2010 (windows 64bit only, 64-bit plguin build)
- `prepcodeblocks.sh` - Unix derivative (linux, bsd, etc); Generates project files for Code::Blocks and makefiles.
- `prepeclipse.sh` - Unix derivatives (linux, bsd, etc); Generates project files for Eclipse and makefiles. will not work correctly on Mac.
- `prepmake.sh` - Unix derivatives (linux, bsd, etc); Generates makefiles.
- `prepmac.sh` - XCode project files, needed to build a plugin on Mac OS X.

* Note that the x64 windows build commands should generally not be used unless you **know** that you need them. Most windows browsers run in 32 bit mode, not 64 bit mode, so a 64 bit build is most likely not what you want. Also note that if you create a 64 bit windows plugin it will only work on 64 bit versions of the browser; firefox is currently 32 bit on windows and the default Internet Explorer choice is also 32 bit, and the 64 bit plugin **will not work** on these browsers.

After you have used a prep script to generate project files of one type, you cannot generate project files for another type without deleting the `build/` directory.

Location of project files

After running a prep command, the project files will be in the `build/` directory.

- On Windows, you'll usually use `build/FireBreath.sln` as your main project
- On Unix derivatives, you'll usually run `make` in the `build/` directory with the name of your project (e.g. `make FBTestPlugin`
- On Mac, you'll usually use `build/FireBreath.xcodeProj`

Location of user-created plugins

When you run a prep command, it scans the `projects/` subdirectory of the FireBreath tree. This directory is not in source control – you have to create it, or `fbgen` will create it for you (see CreatingANewPluginProject). Any directories in there that contain a CMake (plugin) project will be added.

Building example projects

If you call any of the above prep commands with "examples" as an argument (e.g. `prep2008.cmd examples`), project files will be generated in buildex/ and it will use plugins found in the "examples/" subdirectory.

Making changes to your project

When using cmake, you should never make changes directly to the project files (i.e. `FireBreath.sln`, `FBTestPlugin.vcproj`, `Makefile`, etc). Instead, modify one of the following files in your plugin directory. The relevant files in the FBTestPlugin directory are:

- `examples/`
    - `FBTestPlugin/`
        - `CMakeLists.txt`
        - `Win/`
            - `projectDef.cmake`
        - `Mac/`
            - `projectDef.cmake`
        - `Linux/`
            - `projectDef.cmake`

Once changes are made, the "prep" command should be run again so that cmake can generate the updated project files.

If you ever screw up your project file (it happens) and possibly get very weird build errors you cannot explain, simply delete your build directory and regenerate it with the prep script.

CMake uses absolute paths to generate the project files, so if you move the FireBreath tree you will need to re-run your "prep" command.

Problems solved using cmake

With all of that said, many of you may be wondering "Why go through all of this hassle?". Here are a few of the things that CMake makes possible for us:

- There is only one set of project definitions, but with cmake we can build on the following systems (adding others is in many cases just a question of creating a new prep script):
    - Visual Studio 2005, 2008, and 2010
    - GCC/Make on any unix
    - XCode 3.1.x or 3.2.x on Mac OS X
- New plugins added to the `projects/` subdirectory can be automatically detected
- The `ActiveXPlugin` and `NpapiPlugin` projects depend on plugin-specific constants and so need to be built for each plugin; with cmake, you never have to modify these files
- A single file in the plugin directory `PluginConfig.cmake`can be used to generate resource files, registry files, and header files for common plugin configuration items, such as:
    - GUIDs needed by ActiveX that must be unique for each plugin
    - Mimetype of the plugin
        - This item is particularly important because every browser and OS combination configures this in a different way
    - Metadata of the plugin (name, description, version, etc)
    - Other project configuration options can be added for overriding the defaults in projects that you don't want to mess with.

These are just a few of the reasons; if anyone can find a way that works as well or better than cmake at solving all of these problems, we would be happy to consider other options.