



Primordial Soup Features and Outline

So, what is Primordial Soup? Read on and you will find out...

1. Outline

Primordial Soup is a Java based Automatic Build System which is accompanied by a JavaME and Applet Game Engine, called Protein (Note that while the Engine is optimised for creating games, it is equally adept at creating Applications. However, more features will be added in future versions to make creating Applications easier).

Primordial Soup takes the form of several modules, each of which handles a different aspect of the build process and is configured using a series of XML files:

1. Thesaurus – Text Resources
2. Nessie – Sprites, Images and Icons
3. Fontosaurus – System and Custom Fonts
4. Binosaur – All other resources (which can be stored as a simple binary file)
5. Carbon – Handles the preprocessing, compiling, obfuscation and preverification (if necessary) on all source files.
6. Egg – Gathers up all the output from the other modules and creates the final JAR, along with the creation of the JAD file (for JavaME) or an HTML file (for Applets).

Primordial Soup also comes with a flexible preprocessor. The preprocessor is mainly used with the engine source files, but is also used within the XML configuration files. The preprocessor supports the use of brackets and all bitwise operations (including &, !, | and ==) Please see each of the sections for more details on how the preprocessor is used.

Each of the modules generates a series of constants that are used to identify each of the resources that have been gathered and processed by the modules. Feeding these constants into the loading mechanisms of Protein will then load the appropriate resource.

All resources use a consecutive Binary Coded Decimal (BCD) naming system. This means that unless you know what the files are in the JAR then they are useless. Security of your IP and Resources is a big factor for most Development Studios and Primordial Soup includes this BCD system, along with automatic Obfuscation of source code (which can be disabled), to make it even harder for anyone trying to hack or copy your Game or Application.

1.1 Supported Devices

Each of these devices are supported, providing that you have the required SDKs:

Nokia Series 60 DP 1.0	Samsung MIDP 2.0
Nokia Series 60 DP 2.0	Sharp GX10(i), GX15, GX20 and GX30
Nokia Series 40 DP 1.0	Motorola V Series
Nokia Series 40 DP 2.0	Applets
Nokia Series 30 DP 1.0	Generic MIDP 1.0 *
SonyEricsson MIDP 2.0	Generic MIDP 2.0 *

* Note that these are not guaranteed to work on every device.

2. XML Files

Primordial Soup is completely configured by XML files. There are several XML Files used (most of which will not need to be edited for standard use of Primordial Soup).

- *Configuration.xml* – This file configures some of the system properties, including the location of the Obfuscator, JAD/MANIFEST contents and the location of the Protein source code.
- *DeviceProfiles.xml* – This file contains the properties for each of the supported Devices (see above for a list).
- *APIs.xml* – This goes with the DeviceProfiles.xml file and defines the location and some other properties of each API.
- *Modules.xml* – Configuration for the various Modules (see section 2). User defined Modules can also be added in a separate, Project specific XML file.
- *Project.xml* – This is information about the current Project and can be used to override some of the Configuration.xml file (can actually be named anything).

3. Primordial Soup Modules

3.1 Thesaurus

The Thesaurus Module handles the compilation of all Text Resources. Thesaurus can handle multiple language builds, as well as single language builds. All you need to do is specify language packs, or individual languages in the Project Configuration XML file and then, on the command line, pass in the pack or language you wish to use.

An example Language XML file might look like this:

```
<?xml version='1.0' encoding='UTF-16'?>
<Text>
  <Section ID="Main Menu">
    <Entry>SELECT</Entry>
    <Entry>BACK</Entry>
    <Entry>INPUT</Entry>
    <Entry>SPRITES</Entry>
    <Entry>OPTIONS</Entry>
  </Section>
  <Section ID="General">
    <Entry>YES</Entry>
    <Entry>NO</Entry>
    <Entry>EXIT</Entry>
  </Section>
  <Section ID="Input">
    <Entry>CONTINUOUS</Entry>
    <Entry>REPEAT</Entry>
  </Section>
</Text>
```

As you can see, the Text is split up into different Sections. This allows you to only load whichever sections you need, using the `com.protein.resources.Text` class in Protein. When loading the sections you require, you can also specify which language you wish to use. Note that while this example doesn't use multiple lined text, Thesaurus does support it.

3.2 Nessie

Nessie is the Module that takes care of all PNG Resources. There are three kinds of Resources that Nessie can handle:

- Images – Just a standard PNG Image.
- Icon – This is the Icon that you see on the Phone's Game Menu.
- Sprites – Protein contains its own Sprite class, which can handle multiple frames and varying animation sequences (see section 4 for more details).

Images and Sprites are stripped down to the component parts and stored in BCD encoded files (a further step in preventing your resources being hacked). The individual PNG Image Data and Palette are stored in separate files and reconstructed in memory using the `com.protein.resources.ResourceLoader` and/or the `com.protein.game.Sprite` class.

This data is stored uncompressed and while you might think that this would produce a larger JAR size - it doesn't! It actually produces a smaller JAR size. The reason behind this is that the PNG file format for the compressed Image Data is almost identical to that used by the JAR compression program, so, when a PNG is stored in the JAR, it is simply stored, which uses up a small amount of extra data, making them bigger!

Multiple Images and Sprites can also be packed into single BCD files, giving even greater compression. This can also help with loading times as Protein will buffer any BCD files that it loads, so, if you load all of the Images or Sprites from one BCD file in a row, it will only ever have to load the BCD file once!

3.3 Fontosaurus

Fontosaurus handles the definitions of System and Custom Fonts. These fonts are then used when drawing Strings to the canvas. Protein allows you to easily switch between a System and Custom font. You can also dynamically change the colours of both then System and the Custom Fonts.

3.4 Binosaur

Binosaur takes care of all those files that don't fit into any other category. These files are also BCD encoded and can also be grouped just like the Sprites and Images.

3.5 Carbon

Carbon handles the creation of .class files from your .java source. All of your source code is automatically preprocessed, compiled and preverified (JavaME only).

3.6 Egg

Finally, Egg takes up the resources generated by the other Modules and adds them to the JAR, creating the MANIFEST.MF file at the same time. Then it will either create the associated JAD or HTML file (depending on whether you are building a JavaME or Applet version).

4. Protein

Protein is the Game Engine that comes as part of the Primordial Soup Package. There are various classes, each of which handles a different part of a Game or Application. Protein is arranged in a series of packages, this helps to keep the engine organised.

Protein is designed so that it is entirely abstracted from your project's code. Not only does this make it easier to upgrade, it also means that you only need to concentrate on getting your Game or Application coded. Protein's source files are heavily preprocessed in order to take care of the various Device inconsistencies and bugs, letting you get on with being creative!

In order to be totally abstract, Protein runs on a series of Modules. In order to run a Module you simply implement the `com.protein.CanvasModule` class and register the module with the `com.protein.Canvas` class, then fill in the blanks!

Here is the structure of Protein:

- **com.protein.application.BasicApp** – This class deals with the startup and shutdown of the Game or Application. You simply have to extend this class, override the initialisation method and away you go!
- **com.communication.Bluetooth** – This simple class contains all that is needed to set up a Bluetooth connection with another device, including service discovery.
- **com.protein.game.Menu** – This is a simple Game Menu class and is ideal for setting up basic Game Menus that will adjust to the various screen sizes. This class is always being improved upon and will eventually recreate all UI components that you can use on Forms and Lists.
- **com.protein.game.Sprite** – An advanced Sprite class that allows you various degrees of control over the animation sequence, as well as providing Rotation and Translation on those devices that support it.
- **com.protein.io.SavedData** – Handles the saving of any Data that you need to keep between sessions (currently only supported on JavaME devices).
- **com.protein.math.FixedPoint** – This class contains some useful methods for handling Fixed Point Maths, including Sine and Cosine methods.
- **com.protein.math.Point2D** – Allows you to define a point in 2D space as well as a destination for that point. The class then uses fixed point maths and the time given to complete to move from the starting point to the target point. You can also specify different types of movement.
- **com.protein.media.Audio** – A basic audio class for playing individual MIDI files. This class is still under development and will cover all forms of supported Media playback (and recording, if applicable).
- **com.protein.physics.ParticleSystem** – A simple particle System which allows you to define Sprites to use as the emitter and as the particles, with varying methods for handling the way particles are emitted and their behaviour once activated.
- **com.protein.resources.Fonts** – Creates all of the System and Custom Fonts that were defined in your project's XML file.
- **com.protein.resources.ResourceLoader** – Use this class for loading of the Resources generated by some of the Primordial Soup Modules.
- **com.protein.resources.Text** – This class is used to hold the loaded sections of Text that you need, in whatever language is included in the build.
- **com.protein.resources.Resource** – This class is procedurally generated during the build process and contains all of the Resource constants generated by each module.
- **com.protein.special.ClaymoreSplash** – This class is a special class which allows a special Claymore Games splash screen to be included in the application.
- **com.protein.Canvas** – This handles the general running of the Game or Application, including the currently registered com.protein.CanvasModule. It also handles some Debugging functionality (including a profiler, memory meter and FPS counter).
- **com.protein.CanvasModule** – This is an interface that all Modules should implement in order to work with the com.protein.Canvas class.
- **com.protein.Graphics** – This class overrides most of the drawing related routines and uses preprocessing to adjust for various device idiosyncrasies. Also included (for supported devices) is the ability to scale Sprites, using either a basic pixel grab or a bicubic interpolation method. Later there will be other features added, like a Global Transform.

Primordial Soup is constantly being updated and expanded to support new features and new devices. All of Protein's classes are full documented using the JavaDoc system and contains all of the information necessary to get the most out of each and every class.

With all of these features combined you will be able to squeeze more than you ever thought possible into a JAR and do it in a shorter time period!

If you have any questions or comments, feel free to contact me at:

brian@claymoregames.co.uk