By now, we are pretty familiar with TinyOS and nesC, however the MoteWorks getting started guide does provide some additional definitions and clarifications of previous definitions. In addition, it covers some of the more practical aspects of getting an application up and running in nesC and MoteWorks.

First, the primary concepts behind TinyOS are the application, component, module, configuration and interface. An application is a set of components linked together to form a run-time executable. The basic building block of nesC applications are components, which are broken up into two different subtypes. The first subtype are modules, which is simple a component that implements one or more interfaces. The second type are configuration components which "wires" other components together. This way, modules can be written independently from each other and wired together using this type of component, improving readability when examining how an application is wired together vs how it is implemented. In addition, it makes it easy to bring a previously written module from another application and simply "wire" it together with a new configuration. Finally, interfaces are bidirectional, they should specify both commands that a module must implement as well as events that modules must handle.

An application is made up of two Makefiles and nesC files that implement and wire up the application. Makefile and Makefile.component define the dependencies for an application. Makefile tends to have the same contents across all applications, while Makefile.component can be used to specify dependencies for this particular application. The nesC files can be identified because they use the extension ".nc" for all source files—interfaces, modules, and configurations. Comments inside these files can either be single line "//" style comments, or multiline "/* */" style comments. One additional feature available when writing comments is the ability to start a multiline comment with two stars "/** */" and thus inform automatic documentation utilities such as GraphViz to use this comment when generating documentation.

Modules are nesC files that perform two main functions: define the interfaces this module provides, implement those interfaces with nesC code.

```
module ModuleName {
        provides {
                interface StdControl;
        }
}
implementation {
        command result_t StdControl.init() {
                return SUCCESS;
        }
        command result_t StdControl.start() {
                return SUCCESS;
        }
        command result_t StdControl.stop() {
                return SUCCESS;
        } }
```

In the example above, the module is providing the StdControl interface, and provides a simple implementation that simply returns SUCCESS when each function is called, but the module doesn't actually perform any actions. The interfaces provided and implementation are separated into a provides and implementation block respectively.

A configuration can also provide interfaces by wiring components together into more complex interface providers, but it is not required to. In the implementation section of an application configuration, the modules are wired together. For example:

```
Main.StdControl -> MyAppM.StdControl;
```

This code snippet simply tells the compiler that the Main.StdControl interface is provided for by the StdControl interface in MyAppM. This is the most basic wiring that can be provided.

Finally, all applications must have a "Main" component which is the scheduler, or driver, of the application. All nesC application execution starts in this component, and so it must be properly wired into the application with the application configuration.