Now that you've installed Blink, let's look at how it works. Blink, like all TinyOS code, is written in nesC, which is C with some additional language features for components and concurrency.

A nesC application consists of one or more *components* assembled, or *wired*, to form an application executable. Components define two scopes:

one for their specification which contains the names of their *interfaces*,

and a second scope for their implementation.

A component *provides* and *uses* interfaces. The provided interfaces are intended to represent the functionality that the component provides to its user in its specification; the used interfaces represent the functionality the component needs to perform its job in its implementation.

Interfaces are bidirectional: they specify a set of *commands*, which are functions to be implemented by the interface's provider, and a set of *events*, which are functions to be implemented by the interface's user. For a component to call the commands in an interface, it must implement the events of that interface. A single component may use or provide multiple interfaces and multiple instances of the same interface.

The set of interfaces which a component provides together with the set of interfaces that a component uses is considered that component's *signature*.

Configuration and Modules

There are two types of components in nesC: *modules* and *configurations*.

Modules provide the implementations of one or more interfaces.

Configurations are used to assemble other components together, connecting interfaces used by components to interfaces provided by others.

Every nesC application is described by a top-level configuration that wires together the components inside.

Blink: An Example Application

In **Lab 0** you used the Blink Example app to verify TinyOS installation on machine. If we compiled and ran the blink application on actual mote - it displays a counter on the three mote LEDs. In actuality, it simply causes the LED0 to to turn on and off at 4Hz, LED1 to turn on and off at 2Hz, and LED2 to turn on and off at 1Hz. The effect is as if the three LEDs were displaying a binary count of zero to seven every two seconds.

But because we are not working on actual hardware we used this app to fix/verify the TinyOS installation.

Blink is composed of two **components**: a **module**, called "BlinkC.nc", and a **configuration**, called "BlinkAppC.nc". Remember that all applications require a top-level configuration file, which is typically named after the application itself. In this case BlinkAppC.nc is the configuration for the Blink application and the source file that the nesC compiler uses to generate an executable file. BlinkC.nc, on the other hand, actually provides the *implementation* of the Blink application. As you might guess, BlinkAppC.nc is used to wire the BlinkC.nc module to other components that the Blink application requires.

The reason for the distinction between modules and configurations is to allow a system designer to build applications out of existing implementations. For example, a designer could provide a configuration that simply wires together one or more modules, none of which she actually designed. Likewise, another developer can provide a new set of library modules that can be used in a range of applications.

Sometimes (as is the case with BlinkAppC and BlinkC) you will have a configuration and a module that go together. When this is the case, the convention used in the TinyOS source tree is:

File Name	File Type
`Foo.nc`	Interface
`Foo.h`	Header File
`FooC.nc`	Public Module
`FooP.nc`	Private Module

The BlinkAppC.nc Configuration

The nesC compiler compiles a nesC application when given the file containing the top-level configuration. Typical TinyOS applications come with a standard Makefile that allows platform selection and invokes ncc with appropriate options on the application's top-level configuration.

Let's look at BlinkAppC.nc, the configuration for this application first:

```
configuration BlinkAppC {
}
implementation {
  components MainC, BlinkC, LedsC;
  components new TimerMilliC() as Timer0;
  components new TimerMilliC() as Timer1;
  components new TimerMilliC() as Timer2;

  BlinkC -> MainC.Boot;
```

```
  BlinkC.Timer0 -> Timer0;
  BlinkC.Timer1 -> Timer1;
  BlinkC.Timer2 -> Timer2;
  BlinkC.Leds -> LedsC;
}
```

The first thing to notice is the key word configuration, which indicates that this is a configuration file. The first two lines,

```
configuration BlinkAppC {
}
```

simply state that this is a configuration called BlinkAppC. Within the empty braces here it is possible to specify uses and provides clauses, as with a module. This is important to keep in mind: a configuration can use and provide interfaces. Said another way, not all configurations are top-level applications.

The actual configuration is implemented within the pair of curly brackets following the key word implementation. The components lines specify the set of components that this configuration references. In this case those components are Main, BlinkC, LedsC, and three instances of a timer component called TimerMilliC which will be referenced as Timer0, Timer1, and Timer2 <ref name="timermillic_footnote">The TimerMilliC component is a *generic component* which means that, unlike non-generic components, it can be instantiated more than once. Generic components can take types and constants as arguments, though in this case TimerMilliC takes none. There are also *generic interfaces*, which take type arguments only. The Timer interface provided by TimerMilliC is a generic interface; its type argument defines the timer's required precision (this prevents programmer from wiring, e.g., microsecond timer users to millisecond timer providers)

The last four lines wire interfaces that the BlinkC component *uses* to interfaces that the TimerMilliC and LedsC components *provide*. To fully understand the semantics of these wirings, it is helpful to look at the BlinkC module's definition and implementation.

The BlinkC.nc Module

```
module BlinkC {
  uses interface Timer<TMilli> as Timer0;
  uses interface Timer<TMilli> as Timer1;
  uses interface Timer<TMilli> as Timer2;
  uses interface Leds;
  uses interface Boot;
```

```
}
implementation
{
  // implementation code omitted
}
```

The first part of the module code states that this is a module called BlinkC and declares the interfaces it provides and uses. The BlinkC module **uses** three instances of the interface Timer<TMilli> using the names Timer0, Timer1 and Timer2 (the <TMilli> syntax simply supplies the generic Timer interface with the required timer precision). Lastly, the BlinkC module also uses the Leds and Boot interfaces. This means that BlinkC may call any command declared in the interfaces it uses and must also implement any events declared in those interfaces.

After reviewing the interfaces used by the BlinkC component, the semantics of the last four lines in BlinkAppC.nc should become clearer. The line BlinkC.Timer0 -> Timer0 wires the three Timer<TMilli> interface used by BlinkC to the Timer<TMilli> interface provided the three TimerMilliC component. The BlinkC.Leds -> LedsC line wires the Leds interface used by the BlinkC component to the Leds interface provided by the LedsC component.

nesC uses arrows to bind interfaces to one another. The right arrow (A->B) as "A wires to B". The left side of the arrow (A) is a user of the interface, while the right side of the arrow (B) is the provider. A full wiring is A.a->B.b, which means "interface a of component A wires to interface b of component B." Naming the interface is important when a component uses or provides multiple instances of the same interface. For example, BlinkC uses three instances of Timer: Timer0, Timer1 and Timer2. When a component only has one instance of an interface, you can elide the interface name.