```
    // ...create and register other clients for interest...

    engine.run();    // Cede control to e's event loop until complete.

    return 0;
}
```

The implementation of `myCallback`, in the example below, is then free to reregister interest in the same event, save the cookie elsewhere to reregister at a later time, or complete its task and let the `CallbackEngine` take care of properly cleaning up all now unnecessary resources:

```
void myCallback(const EventData&     event,
                CallbackEngine*      engine,
                const CallbackData&  cookie)
{
    int status = EventProcessor::processEvent(event);

    if (status > 0)  // Status is nonzero; continue interest in event now.
    {
        engine->reregisterInterest(cookie);
    }
    else if (status < 0)  // Negative status indicates EventProcessor wants
                          // to reregister later.
    {
        EventProcessor::storeCallback(engine, cookie);
                          // Call reregisterInterest later.
    }

    // Return flow of control to the CallbackEngine that invoked this
    // callback.  If status was zero, then this callback should be cleaned
    // up properly with minimal fuss and no leaks.
}
```

What makes use of the opaque enumeration here especially apt is that the internal data structures maintained by the `CallbackEngine` might be subtly interrelated, and any knowledge of a client's relationship to those data structures that can be maintained through callbacks is going to reduce the amount of lookups and synchronization that would be needed to correctly reregister a client without that information. The otherwise wide contract on `reregisterInterest` means that clients have no need themselves to directly know anything about the actual values of the `State` they might be in. More notably, a component like this is likely to be heavily reused across a large codebase, and being able to maintain it while minimizing the need for clients to recompile can be a huge boon to deployment times.

To see what is involved, we can consider the business end of the `CallbackEngine` implementation and an outline of what a single-threaded implementation might involve: