```
    std::deque<CallbackData> d_pendingCallbacks;
        // The collection of clients currently registered for interest, or having
        // callbacks delivered, with this CallbackEngine.
        //
        // Reregistering or skipping reregistering when
        // called back will lead to updating internal data structures based on
        // the current value of this State.

public:
    // ...   (other public member functions, e.g., creators, manipulators)

    void registerInterest(CallbackData::Callback cb);
        // Register (e.g., from main) a new client with this manager object.

    void reregisterInterest(const CallbackData& callback);
        // Reregister (e.g., from a client) the specified callback with this
        // manager object, providing the state contained in the CallbackData
        // to enable resumption from the same state as processing left off.

    void run();
        // Start this object's event loop.

    // ...  (other public member functions, e.g., manipulators, accessors)
};
```

A client would, in `main`, create an instance of this `CallbackEngine`, define the appropriate functions to be invoked when events happen, register interest, and then let the engine `run`:

```
// myapplication.cpp:
// ...
#include <callbackengine.h>

static void myCallback(const EventData&    event,
                       CallbackEngine*     engine,
                       const CallbackData& cookie);
    // Process the specified event, and then potentially reregister the
    // specified cookie for interest in the same data.

int main()
{
   CallbackEngine engine;  // Create a configurable callback engine object.

   //...     (Configure the callback engine~~, e,~~ as appropriate.)

   engine.registerInterest(&myCallback);  // Even a stateless function pointer can
                                          // be used with std::function.
```