How to work with Application::process callbacks

Overview

There are two Application::process callbacks in the FIX Antenna HFT: Application::process(msg, aSn) and Application::process(work, sn), the first one is devoted to work with non-accelerated sessions only so the second one should be preferred. To use Application::process(work, sn), two parameters should be set in the engine.properties: **UseTCPDispatcher = true** and **Session.Default.UseLiteMessage = true**.

Application::process is a callback method to process in-sequence incoming application (business) level messages. If the application can not process the given message in the callback, the message can be *pipelined* / *throttled* / *offloaded or rejected.*

To *pipeline* the message means to set **work->pipelined = 1** and return true. The **work->elem** object with the message is owned by the user application until **elem->returnObj()** is called and can be processed in the separate thread. Call **sn.commitIncomingMessage(work)** before pass **work->elem** to the separate thread if the incoming message should be logged before any reply message.

To *throttle* the message means to return false from the callback. The engine will drop the message and send ResendRequest when sequence gap is detected so it will be delivered again.

To *offload* the message means to set **work->offloadAdminMessage = 1** and return true. The engine will schedule delivering of the message again using the thread from the pool and pause the socket reading until the message is processed.

If an exception is thrown from the callback, the *reject* message will be sent by the engine to the counterparty.

The Application::process method should be completed as quickly as possible because multiple sessions are served by each Reader thread of the new TCPDispatcher. Time-consuming tasks are not advised to be performed inside it, pass the message into the pool to be processed instead.

```
bool TraderApp::process( Engine::FIXMessageProcessElem* work, const Engine::Session& sn )
{
    sn.commitIncomingMessage(work);                // save incoming message to session logs (to cache in
memory)
    work->elem->userData0 = (MyApp*)this;          // setup processing pool to call this app to process the
message
    processor_->postMessageToWorker( work->elem );  // post the messaage to the pool
    work->pipelined = 1;                           // hint the engine that the message is still in use
    return true;
}
```

Thread Safety

The caller of the Application::process() method is Session, It guaranties that only one thread at the same time calls the Application object callbacks **from this session**. So it guaranties thread safety if the Application is bound to one session. The caller thread can be Worker, Reader (in HFT FA), Timer or user thread that manipulates Session. FIX Antenna HFT always uses direct sending from the user threads to the sockets if sockets are ready. Otherwise (if sockets are busy), it uses the old style single dispatcher to poll sockets until a socket gets ready (vacant).

Different sessions can be served by different threads at the same time but the socket reading is performed by one thread per session and the next reading will not start by the other thread until the current thread finishes processing (that includes delivery to the Application object). As a part of the reading process it cannot be called concurrently if the Application is bound to the one Session object.

The socket **accept()** call is on the same thread as the **read()** thread (unlike FIX Antenna C++). After socket is accepted and the Logon message is read by the Reader, the processing of the Logon message is performed by the Worker thread, so if one Application object is bound to the multiple Sessions it can receive **onLogonEvent()** from the one session and **process()** from the another concurrently even if the one Reader thread is used.

The constness of the session ref in the Application::* callbacks means one generally shouldn't call synchronously non-const methods of the session from this callback even if stored non-const session ref is obtained a different way.

Reader Thread

The number of readers is controlled by the property **TCPDispatcher.NumberOfWorkers** in the engine.properties configuration file passed to **FixEngine::init** directly or using **InitParameters::propertiesFileName**. Each Reader has its own pool of messages. Each Reader calls the **onNewThreadCreated()** method with the TCP_READER value in the threadAttrs.