FIX Antenna ConnectToGateway sample

Overview

The sample demonstrates how a FIX Antenna C++ user can connect to the FIX Gateway (venue) and send pre-loaded messages.

The ConnectToGateway sample shows how to use the scheduling functionality described in Scheduler.

The features implemented in the sample:

- Establish a connection with FIX Gateway as FIX initiator and disconnect from it according to the schedule
- Switch to backup connections in case if the primary connection becomes unavailable
- Send a FIX message from the file to the counterparty
- Create a New Order - Single (D) using FIX Antenna API and setting various FIX tags
- Create a session configured in the `engine.properties` configuration file
- Use of a customized Logon (A) message for passing user credentials
- Customized reaction on Resend Request (2) messages
- Print the received messages on a console
- Reset sequence numbers according to the schedule

Application preparation

The application initializes the Engine and prepares a message for sending. Message content can be prepared from the file configured in the `MsgFile` property or via API if the `MsgFile` property is not set.

Configuring the sessions

The ConnectToGateway sample covers the following scenarios:

1. Sessions are not configured in `engine.properties`.
   In this case, the sample starts the hardcoded session.
2. Sessions are configured in `engine.properties` without a schedule.
   In this case, the sample starts the pre-configured session immediately.
3. Sessions are configured in `engine.properties` with a schedule.
   In this case, the sample starts the pre-configured session according to the schedule.
   Sequences are reset on `Ontradeperiodstart`.

A user can pass a customized logon message via the `CustomLogon` property.

Overriding classes

A user should inherit a class from `Engine::Application` and override the virtual methods.

The `MyApp` class in the ConnectToGateway sample: all important callbacks print the information to the console. On resend request, it asks a user for action in case of resend requests processing.

If the user wants to define scheduled actions before a trading day or after, it is needed to override callbacks inherited from the `SessionsScheduleManager` class.

The `MyScheduleManager` class in the sample: Sessions Schedule Manager drops sequence numbers, uses the custom logon at onTradePeriodBegin.

Using schedule in user's application

Scheduling functionality is described in the Programmer's Guide (refer to the Scheduler section).

For applying the schedule, a user should initiate sessions from `engine.properties`.

MyApp.cpp

```
 FAProperties::ConfiguredSessionsMap sessions = FixEngine::singleton()->getProperties()->getConfiguredSessions
(INITIATOR_SESSION_ROLE);
```

The Scheduler calls the function of a schedule. The schedule function prepares the execution plan and provides its execution.

The schedule function interacts with the Session Controller to obtain the list of all sessions which are registered by the Controller.

The schedule receives notifications about `TradePeriodBegin` and `TradePeriodEnd` events.

The schedule accepts events in the cron-like format.

If sessions have a schedule, then the user can pass the session schedule to the Controller. The further session state management will be done by the Session Controller according to the session schedule.

MyApp.cpp

```
pSchedule = Engine::SessionsSchedulePtr(FixEngine::singleton()->getScheduler()->getSchedule(sessionIt->second.
schedule_));
// ...
sessionsController_ = FixEngineSessionsControllerPtr(pSchedule->getSessionsController());
```

Session Controller is an owner of one or multiple sessions in the application.

Each session should be registered with the Session Controller. Otherwise, a session cannot be started according to the schedule.

Session Controller provides the uniform interface to start/stop a session, to perform connection/disconnection of a session via ID.

For resetting sequences at the start of trading day, the Session Controller should be passed to user's Scheduler manager.

MyApp.cpp

```
mySchedulerManager.setController(sessionsController_);
pSchedule->attachScheduleManager(&mySchedulerManager);
```

Sessions Schedule Manager receives notifications about the `TradePeriodBegin` and `TradePeriodEndevent` events and gets the execution plan.

Finally, ownership over the session should be granted to the Sessions Controller:

MyApp.cpp

```
sessionsController_->registerSession(sessionId_, *this, sessionIt->second);
```