Appendix A. Interfaces

Mit::Application

```
// Releasing

virtual void release() const = 0;

// Name

virtual std::string getName() const = 0;

// Opening / closing

virtual bool opened() const = 0;

// Initializes the application, creates and adds services from the configuration file.
// The services are created in the disconnected state and should be connected explicitly.
// applicationListener should stay alive until close() is called.
virtual void open(ApplicationListener* applicationListener) = 0;

// Removes the services and finalizes the application.
// Should only be called if open() succeeded.
virtual void close() = 0;

// Services

// Returns the number of existing services.
virtual size_t getServiceCount() const = 0;

// Returns a service by index.
virtual Service* getService(size_t index) = 0;

// Finds a service by name. Returns 0 if the service does not exist.
virtual Service* findService(const std::string& name) = 0;

// Creates and adds a new service to the application.
// The service is created in the disconnected state and should be connected explicitly.
virtual Mit::Service* addService
(
    ServiceType serviceType,
    const std::string& name,
    const ServiceOptions& serviceOptions
) = 0;

// Removes the service from the application and destroys it.
virtual void removeService(const std::string& name) = 0;

// Removes all the services from the application and destroys them.
virtual void removeAllServices() = 0;
```

Mit::ApplicationListener

```
// Services

// Called after a service is created and added to the application
virtual void onServiceAdded(Application* application, Service* service) = 0;

// Called before a service is removed from the application and destroyed
virtual void onServiceRemoving(Application* application, Service* service) = 0;

// Instruments

// Called after an instrument is created and added to a service
virtual void onInstrumentAdded(Application* application, Service* service, Instrument* instrument) = 0;

// Called before an instrument is removed from a service and destroyed
virtual void onInstrumentRemoving(Application* application, Service* service, Instrument* instrument) = 0;
```

Mit::Service

```
// Releasing

virtual void release() const = 0;

// Type

virtual ServiceType getType() const = 0;

// Name

virtual std::string getName() const = 0;

// Connecting / disconnecting

virtual bool connected() const = 0;

// Connects the service to the mutlicast group.
// serviceListener should stay alive until disconnect() is called.
virtual void connect(ServiceListener* serviceListener) = 0;

// Disconnects the service from the mutlicast group.
// Should only be called if connect() succeeded.
virtual void disconnect() = 0;

// Instruments

// Returns the number of existing instruments.
virtual size_t getInstrumentCount() const = 0;

// Returns an instrument by index.
virtual Instrument* getInstrument(size_t index) = 0;

// Finds an instrument by ID. Returns 0 if the instrument does not exist.
virtual Instrument* findInstrument(const std::string& id) = 0;

// Creates and adds a new instrument to the service.
virtual Instrument* addInstrument(const std::string& id) = 0;

// Removes the instrument from the service and destroys it.
virtual void removeInstrument(const std::string& id) = 0;

// Removes all the instruments from the service and destroys them.
virtual void removeAllInstruments() = 0;

// Statistics

// Returns service statistics snapshot.
// connectionStatistics can be 0.
// dateStatistics can be 0.
virtual void getStatistics
(
    ServiceConnectionStatistics* connectionStatistics,
    ServiceDateStatistics* dateStatistics
) const = 0;
```

Mit::ServiceListener

```
// Messages

// Called on service reset. The client should reset all its message derived data.
virtual void onReset(Service* service, ResetReason resetReason) = 0;

// Called on FIX message.
virtual void onFixMessage(Service* service, Engine::FIXMessage* message) = 0;

// Called on ITCH message.
virtual void onItchMessage(Service* service, const Jse::Itch::MessageHeader* messageHeader) = 0;

// Called on ITCH message.
virtual void onItchMessage(Service* service, const Lse::Itch::MessageHeader* messageHeader) = 0;
```

Mit::Instrument

```
// Releasing

virtual void release() const = 0;

// ID

virtual std::string getId() const = 0;

// Subscribing / unsubscribing

virtual bool subscribed() const = 0;

// Subscribes to an instrument.
// instrumentListener should stay alive until unsubscribe() is called.
virtual void subscribe(InstrumentListener* instrumentListener) = 0;

// Unsubscribes from an instrument.
// Should only be called if subscribe() succeeded.
virtual void unsubscribe() = 0;

// Returns instrument statistics snapshot.
virtual void getStatistics(InstrumentStatistics& statistics) = 0;
```

Mit::InstrumentListener

```
// Messages

// Called on an instrument reset. The client should reset all its message derived data.
virtual void onReset(Instrument* instrument, ResetReason resetReason) = 0;

// Called on a FIX message.
virtual void onFixMessage(Instrument* instrument, Engine::FIXMessage* message, PerformanceIndicator*
performanceIndicator) = 0;
```

Mit::ConnectionManager

```
// Releasing

virtual void release() const = 0;

// Opening / closing

virtual bool opened() const = 0;

// Initializes connection manager
virtual void open() = 0;

// Finalizes connection manager
virtual void close() = 0;

// Connections

// Creates a UDP connection.
// The connection is created in the disconnected state and should be connected explicitly.
// The connection should be destroyed with UdpConnection::release().
// It is possible to have several connections created at the same time.
virtual UdpConnection* createUdpConnection(const std::string& name, bool logData) = 0;

// Creates a TCP connection.
// The connection is created in the disconnected state and should be connected explicitly.
// The connection should be destroyed with TcpConnection::release().
// It is possible to have several connections created at the same time.
virtual TcpConnection* createTcpConnection(const std::string& name, bool logData) = 0;

// Tasks

// Posts the task for asynchronous execution in the context of another thread.
// connectionManagerListener should stay alive until ConnectionManagerListener::onTask() is called.
virtual void postTask
(
    const std::string& name,
    void* parameters,
    ConnectionManagerListener* connectionManagerListener
) = 0;
```

Mit::UdpConnection

```
// Releasing

virtual void release() const = 0;

// Name

virtual std::string getName() const = 0;

// Connecting / disconnecting

virtual bool connected() const = 0;

// Connects to the multicast group.
// connectionListener should stay alive until disconnect() is called.
virtual void connect
(
    const std::string& localAddress,
    const std::string& groupAddress,
    System::u16 port,
    UdpConnectionListener* connectionListener
) = 0;

// Disconnects from the multicast group.
// Should only be called if connect() succeeded.
virtual void disconnect() = 0;
```

Mit::UdpConnectionListener

```
// Data

// Returns the size of data to receive and pass to onDataReceived() or 0.
virtual size_t getDataSize(UdpConnection* connection, const void* data, size_t size) = 0;

// Called on data received.
virtual void onDataReceived(UdpConnection* connection, const void* data, size_t size) = 0;

// Called to notify that no more data is available.
virtual void onNoData(UdpConnection* connection) = 0;
```

Mit::TcpConnection

```
// Releasing

virtual void release() const = 0;

// Name

virtual std::string getName() const = 0;

// Connecting / disconnecting

virtual bool connected() const = 0;

// Connects to the remote endpoint.
virtual void connect(const std::string& localAddress, const std::string& remoteAddress, System::u16 port) = 0;

// Disconnects from the remote endpoint.
// Should only be called if connect() succeeded.
virtual void disconnect() = 0;

// Sending / receiving

// Sends the data synchronously.
virtual void sendData(const void* data, size_t size) = 0;

// Receives the size of data synchronously.
virtual void receiveData(void* data, size_t size) = 0;
```