llfix

Low latency FIX engine

Version 1.0.0 • Jan 2026

Chapter 1

llfix Manual

llfix is a low-latency FIX protocol engine developed by Coreware Limited. This manual applies to both the open-source (MIT-licensed) edition and the commercial edition of llfix.

llfix is a product of Coreware Limited.

1.1 Introduction

llfix is a high-performance, low-latency FIX protocol engine designed for latency-sensitive systems. It focuses on deterministic performance, minimal overhead, achieving single-digit microsecond message encoding and decoding without memory allocations on latency-critical paths.

llfix is header-only, written in modern C++17, and supports both Linux and Windows, making it easy to integrate into existing infrastructures. The engine operates exclusively in immediate mode, avoids internal message queueing, and provides optimised time access, fast virtual-memory-based serialisation, and built-in operational tooling. llfix is suitable for environments where predictability, transparency, and performance are critical, from colocated trading systems to high-throughput FIX gateways.

1.2 How to build

llfix can be built using GCC, Clang, or MSVC. The compiler must support C++17. It can be built either in header-only mode or as a static library.

For header-only usage, add the <llfix_root>/include directory to your compiler's include path.

For static library usage, build the library using CMake in the <llfix_root>/lib directory. Then add the <llfix_root>/include directory to your compiler's include path and link the llfix static library in your linker settings. Available CMake options are described in the CMake Support section. When using the static library build, always include llfix/common.h or llfix/engine.h to avoid potential ODR or ABI inconsistencies.

On Linux, you must also link with -pthread.

Windows note: Always include llfix/engine.h before any Windows networking headers. This avoids conflicts between ws2tcpip.h and windows.h, and ensures that the maximum number of socket descriptors that can be monitored is configured correctly.

1.2.1 Building with dependencies

The following optional components can be enabled at build time. Each dependency requires specific preprocessor definitions and build and runtime configuration, as detailed below.

All define requirements listed below must be set at the build level (via compiler settings or target_compile_definitions), not inside individual source files, to avoid ODR/ABI mismatches.

Tinyxml2

Requirement	Description
Defines	-DLLFIX_ENABLE_DICTIONARY
Includes	<llfix_root_directory>/deps/tinyxml2/include/ , provided in the commercial edition repo
Linker	Lib path <llfix_root_directory>/deps/tinyxml2/lib and -ltinyxml2 on Linux, tinyxml2.lib on Windows, both provided in the commercial edition repo
Runtime	N/A

Note: The provided Linux static libraries are built against RHEL 9.4's C++ runtime. To build a static library for your distribution, use deps/tinyxml2/CMakeLists.txt :

```
cd  <llfix_root_directory>/deps/tinyxml2
mkdir build
cd build
cmake ..
make
```

On Windows, the provided libs are built with /MD (Release) and /MDd (Debug). If your project uses /MT or /MTd, change the runtime library setting in the generated Visual Studio project and rebuild tinyxml2.

LibNUMA (Linux only)

Requirement	Description
Defines	-DLLFIX_ENABLE_NUMA
Includes	Libnuma headers must be available on the build host
Linker	-lnuma
Runtime	so file should be available on the runtime host

OpenSSL (version 3.6 or newer)

Requirement	Description
Defines	-DLLFIX_ENABLE_OPENSSL
Includes	OpenSSL headers must be available on the build host
Linker	-lssl -lcrypto on Linux, libcrypto.lib libssl.lib required on Windows
Runtime	so/dll files should be available on the runtime host

As an additional step on Windows, you must add OpenSSL's applink.c to your project sources and build it as part of the application.

Solarflare TCPDirect (Linux only)

Requirement	Description
Defines	-DLLFIX_ENABLE_TCPDIRECT
Includes	TCPDirect headers must be available on the build host
Linker	-lonload_zf_static
Runtime	A functional Solarflare Onload is required at runtime. This dependency applies regardless of whether the application is Onload'ed or uses TCPDirect.

1.2.2 CMake support

You can also find llfix.cmake in the llfix root directory, which can be used as a stand alone helper CMake file for integrating llfix into projects that use CMake. The minimum required CMake version is 3.8. The examples in the examples directory make use of this file.

The CMake options you can pass to llfix.cmake :

Option	Description
-DLLFIX_STATIC_LIB=ON	llfix as a static library (Default mode is header-only)
-DLLFIX_ENABLE_NUMA=ON	LibNUMA
-DLLFIX_ENABLE_DICTIONARY=ON	FIX dictionary support
-DLLFIX_ENABLE_OPENSSL=ON	OpenSSL support
-DLLFIX_ENABLE_TCPDIRECT=ON	Solarflare TCPDirect support

As for OpenSSL option, if the OPENSSL_ROOT_DIR environment variable is not set, llfix will attempt to include and link against the system-provided OpenSSL installation. If it is set, both include paths and libraries will be taken from that directory.

The OPENSSL_ROOT_DIR directory must contain both an include and a lib folder on Linux and Windows. The lib folder must contain the following files:

- On Linux: libssl.so, libcrypto.so
- On Windows: libssl.lib, libcrypto.lib

Additionally on Windows, it must contain "ms" folder with the following OpenSSL file: applink.c .

1.3 Configs

llfix uses a simple, human-readable configuration file format based on INI files similar to Quickfix :

- Lines starting with # are comments
- Configuration entries are organized into groups using square brackets. Group names should be unique: [GROUP_NAME] key=value

There are 4 config categories :

- Engine : is the single application instance. It can manage multiple FIX clients/servers and abstracts all common and central settings.
- FixClient : represents a FIX client/connector. Configs define network settings, retry behaviour, SSL, and NIC bindings for outgoing sessions.
- FixServer : represents a FIX server/acceptor. Configs define accept ports, worker threads, NIC settings, SSL, and connection handling for incoming sessions.
- FixSession : represents an individual FIX session between a client and server. Configs cover protocol version, heartbeat, logon/logoff behaviour, message validations, message serialisation, resends, throttling, scheduling, and dictionary settings.

Example configurations can be found at:

- examples/fix_trade_client/config.cfg – FIX client configuration
- examples/fix_exchange_simulator/config.cfg – FIX server configuration

Both configurations must define an ENGINE group.

Instance names must be unique. For FIX servers, session names must begin with 'SESSION' and be unique.

All loaded configuration values can be inspected via the log files or the admin GUI.

1.3.1 Engine configs

Config	Description	Default
log_level	Logging verbosity level. One of : ERROR WARNING INFO DEBUG	INFO
log_file	Path to the log output file	log.txt
ignore_sighup	Ignore SIGHUP signal, affects the entire process (Linux only)	false
management_server_port	Port for TCP admin interface	0
management_server_nic_ip	NIC IP for TCP admin interface	
management_server_cpu_core↩ _id	CPU core to pin the admin interface thread	-1
numa_bind_node	NUMA node to bind, affects the entire process (Linux only)	-1
numa_aware_allocations	Used together with numa_bind_node. Setup allocations will be from the bound NUMA node (Linux only)	false
lock_pages	Lock the process virtual memory pages using mlockall (Linux only)	false

1.3.2 FIX Client configs

Config	Description	Default
cpu_core_id	CPU core to pin the FIX client thread	-1
nic_address	NIC card IP address	
nic_name	NIC card name. Mandatory for Solarflare TCPDirect	
primary_address	Target endpoint IP address	
primary_port	Target endpoint port no	
secondary_address	Secondary target endpoint IP address	
secondary_port	Secondary target endpoint port no	
connect_timeout_seconds	Connection timeout in seconds (Solarflare TCPDirect only)	3
nic_ringbuffer_tx_size	NIC transmit ring buffer size (Solarflare TCPDirect only)	2048
nic_ringbuffer_rx_size	NIC receive ring buffer size (Solarflare TCPDirect only)	4096
send_try_count	Number of send retry attempts on failure. 0 means infinite retries.	0
starts_as_primary_instance	Start as a primary instance. false value will disable the session.	true
refresh_resend_cache_during_promotion	When a secondary instance is promoted to primary, refreshes the resend message cache.	true
socket_rx_size	Socket receive buffer size in bytes	212992
socket_tx_size	Socket transmit buffer size in bytes	212992
rx_buffer_capacity	Internal receive buffer capacity in bytes	212992
tx_encode_buffer_capacity	Internal transmit buffer capacity in bytes	212992
disable_nagle	Disable Nagle's algorithm	true
quick_ack	Enable TCP quick acknowledgment	false
receive_size	Maximum bytes read per receive call	4096
async_io_timeout_nanoseconds	Async I/O polling timeout in nanoseconds	1000
use_ssl	Enable SSL/TLS	false
ssl_version	SSL/TLS protocol version. One of TLS10 TLS11 TLS12 TLS13	TLS12
ssl_cipher_suite	Allowed SSL/TLS cipher suites in OpenSSL format	
ssl_certificate_pem_file	SSL certificate file in PEM format	

Config	Description	Default
ssl_private_key_pem_file	SSL private key file in PEM format	
ssl_private_key_password	Password for SSL private key file	
ssl_ca_pem_file	CA certificate file in PEM format	
ssl_verify_peer	Verify peer certificate during SSL handshake	true

1.3.3 FIX Server configs

Config	Description	Default
cpu_core_id	Applies to singlethreaded FIX server. CPU core to pin its thread	-1
worker_thread_count	Applies to multithreaded FIX server. If not specified it will be no of physical CPU cores on the host	
accept_port	Server port used to accept incoming connections	
accept_timeout_seconds	Timeout for accepting new connections in seconds	5
nic_address	NIC ip address	
nic_name	NIC card name	
send_try_count	Number of send retry attempts on failure. 0 means infinite retries.	0
starts_as_primary_instance	Start as a primary instance. false value will disable all sessions.	true
refresh_resend_cache_during_↩ promotion	When a secondary instance is promoted to primary, refreshes the resend message cache.	true
socket_rx_size	Socket receive buffer size in bytes	212992
socket_tx_size	Socket transmit buffer size in bytes	212992
rx_buffer_capacity	Internal receive buffer capacity in bytes	212992
tx_encode_buffer_capacity	Internal transmit buffer capacity in bytes	212992
disable_nagle	Disable Nagle's algorithm	true
quick_ack	Enable TCP quick acknowledgment	false
pending_connection_queue_size	Maximum number of pending TCP connection requests	32
receive_size	Maximum bytes read per receive call	4096
async_io_timeout_nanoseconds	Async I/O polling timeout in nanoseconds	1 million on Linux, 1000 on Windows
max_poll_events	Maximum number of events processed per poll cycle	64
use_ssl	Enable SSL/TLS	false
ssl_version	SSL/TLS protocol version. One of TLS10 TLS11 TLS12 TLS13	TLS12
ssl_cipher_suite	Allowed SSL/TLS cipher suites in OpenSSL format	
ssl_certificate_pem_file	SSL certificate file in PEM format	
ssl_private_key_pem_file	SSL private key file in PEM format	
ssl_private_key_password	Password for SSL private key file	

Config	Description	Default
ssl_ca_pem_file	CA certificate file in PEM format	
ssl_verify_peer	Verify peer certificate during SSL handshake	true
ssl_crl_path	Path to Certificate Revocation List (CRL) files for SSL verification	

1.3.4 FIX Session configs, General

Config	Description	Default
heartbeat_interval_seconds	FIX heartbeat interval in seconds. Applies to FIX clients only.	30
enable_simd_avx2	If enabled, FIX checksum (tag10) encoding and validations will be done using SIMD AVX2	false
timestamp_subseconds_precision	Subsecond precision used for FIX tag52(sending time). Values are : NANO MICRO MILLI NONE	NANO

1.3.5 FIX Session configs Header

Config	Description	Default
begin_string	FIX protocol version. One of FIXT.1.1, FIX.4.4, FIX.4.3, FIX.4.2, FIX.4.1, FIX.4.0	
sender_comp_id	SenderCompID identifying this side of the FIX session	
target_comp_id	TargetCompID identifying the counterparty FIX session	
additional_static_header_tags	Additional static FIX tag value pairs in the following comma separated format : <tag>=<value>,<tag>=<value>,...	
include_last_processed_seqnum_in_header	Includes last processed sequence number in outgoing message headers tag369	false

1.3.6 FIX Session configs, Logon and Logout

Config	Description	Default
default_app_ver_id	Default application version ID sent during logon (tag 1137). Applies to FIX5.0 and later	
logon_username	Username included in the logon message for authentication	
logon_password	Password included in the logon message for authentication	
logon_message_new_password	New password sent during logon for password change workflows	
logon_reset_sequence_numbers	Requests sequence number reset during logon (tag141)	false
logon_include_next_expected_seq_no	Includes next expected sequence number in the logon message (tag 789)	false
logon_timeout_seconds	Time to wait for a logon response before timing out	5
logout_timeout_seconds	Time to wait for logout confirmation before forcefully closing the session	5

1.3.7 FIX Session configs, Validations

Config	Description	Default
validations_enabled	Enables or disables all FIX message validations	true
dictionary_validations_enabled	Enables validation of messages against the loaded FIX dictionaries	true
max_allowed_message_age_seconds	Maximum allowed age of incoming messages in seconds based on tag52(sending time). 0 disables check	0
validate_repeating_groups	Enables validation of repeating groups	false

1.3.8 FIX Session configs, Resend Requests

Config	Description	Default
replay_messages_on_incoming_resend_↩ request	If turned on message replaying will be active. If false, gap filling will be used	false
replay_message_cache_initial_size	Initial capacity of the message cache used for handling resends	10240
max_resend_range	Maximum number of messages that can be requested in a single resend request	10000
include_t97_during_resends	Include PossDupFlag (tag 97) on messages resent due to a resend request	false
outgoing_resend_request_expire_in_secs	Time in seconds after which an outgoing resend request is considered expired	30

1.3.9 FIX Session configs, Throttler

Config	Description	Default
throttle_window_in_milliseconds	Time window in milliseconds used to measure message rate for throttling	1000
throttle_limit	Maximum number of messages allowed within the throttle window (0 disables)	0
throttle_action	Action to take when the throttle limit is exceeded. One of WAIT DISCONNECT R↩ EJECT (FixServer only)	WAIT
throttler_reject_message	Message text returned (tag58) when a message is rejected due to throttling (Fix↩ Server only)	"Message rate limit exceeded"

1.3.10 FIX Session configs, Serialisation

Config	Description	Default
sequence_store_file_path	File path where FIX sequence numbers are persisted	sequence.store
incoming_message_serialisation_path	Directory path for storing serialised incoming FIX messages	messages_incoming
outgoing_message_serialisation_path	Directory path for storing serialised outgoing FIX messages	messages_outgoing
max_serialised_file_size	Maximum size (in bytes) of a serialised message file before rotation. 0 disables message serialisation	67108864 (64MB)

1.3.11 FIX Session configs, Scheduler

Config	Description	Default
schedule_week_days	Days of the week when the FIX session is allowed to run. Should be dash ('-') separated digits from 1(Monday) to 7(Sunday)	
start_hour_utc	UTC hour when the session becomes active (-1 disables scheduling)	-1
start_minute_utc	UTC minute when the session becomes active. (-1 disables scheduling)	-1
end_hour_utc	UTC hour when the session stops being active. (-1 disables scheduling)	-1
end_minute_utc	UTC minute when the session stops being active (-1 disables scheduling)	-1

1.3.12 FIX Session configs, Dictionary

Config	Description	Default
application_dictionary_path	File path to the FIX XML dictionary	
transport_dictionary_path	File path to the FIX XML transport/session dictionary. Applies to FIX5.0 and later	

1.4 Code generator

The llfix code generator takes FIX dictionaries as input and generates source code, including:

- Enums for all field values
- Classes per message type with encode and decode methods

The generator is located in the tools/code_generator directory. To build it:

- make release on Linux
- run build_msvc.bat or use VisualStudio project file on Windows.

CLI options are :

Option	Description
-n <namespace>	Namespace name
-d <dict_file>	FIX dictionary file path
-t <transport_dict_file>	FIX Transport dictionary file path
-o <output_path>	Output path
-c <client class name>	Class name for your FIX client
-x <client transport name>	One of TCPConnector TCPConnectorSSL TCPConnectorTCPDirect. TCP↩ Connector is the default one.
-s <server class name>	Class name for your FIX server
-y <server transport name>	One of TcpReactorScalable TcpReactorScalableSSL TcpReactor. Tcp↩ ReactorScalable is the default one.

You can find "generated_server" and "generated_client" examples in "examples" directory.

Minimal usage example :

```
#include "messages/NewOrderSingle.h"
#include "messages/ExecutionReport.h"
void send_execution_report(llfix::FixSession* session, const SimulatorExecReport& contents)
{
    custom::FIX44::ExecutionReport execution_report(outgoing_message_instance(session));
    execution_report.set_msg_type(custom::FIX44::MsgType::EXECUTION_REPORT);
    execution_report.set_OrderID(contents.clorid);
    execution_report.set_ClOrdID(contents.clorid);
    send_outgoing_message(session, execution_report.outgoing_message()
}
virtual void on_new_order(llfix::FixSession* session, const llfix::IncomingFixMessage* message) override
{
    custom::FIX44::NewOrderSingle incoming_new_order(message);
    SimulatorExecReport exec_report;
    exec_report.clorid = incoming_new_order.get_ClOrdID();
    exec_report.symbol = incoming_new_order.get_Symbol();
    exec_report.side = incoming_new_order.get_Side();
    ...
}
```

You can find the full usage example in "tests/stress_tests/server" directory.

1.5 Examples

Main examples (exchange simulator and trade client) can be found in the examples directory. To build them :

- make release on Linux and build_msvc.bat or VisualStudio project file on Windows

Alternatively you can also use CMake :

```
mkdir build
cd build
cmake ..
```

The options you can pass to CMake :

Option	Description
-DLLFIX_STATIC_LIB=ON	llfix as a static library (Default mode is header-only)
-DLLFIX_ENABLE_NUMA=ON	LibNUMA
-DLLFIX_ENABLE_DICTIONARY=ON	FIX dictionary support
-DLLFIX_ENABLE_OPENSSL=ON	OpenSSL support
-DLLFIX_ENABLE_TCPDIRECT=ON	Solarflare TCPDirect support

In the exchange simulator example :

- For switching to multithreaded server, you need to include llfix/core/utilities/tcp_reactor_scalable.h and derive ExchangeSimulator class from llfix::TcpReactorScalable<>
- For switching to multithreaded SSL server, you need to include llfix/core/ssl/tcp_reactor_scalable_ssl.h and derive ExchangeSimulator class from llfix::TcpReactorScalableSSL<>

In the trade client example :

- For switching to SSL client , you need to include llfix/core/ssl/tcp_connector_ssl.h and derive SampleClient class from llfix::TCPConnectorSSL
- For switching to Solarflare TCPDirect client , you need to include llfix/core/solarflare_tcpdirect/tcp_↩ connector_tcpdirect.h and derive SampleClient class from llfix::TCPConnectorTCPDirect

You can also find other examples under "tests/other_networked_tests" directory :

- "logon_password_authentication" demonstrates specifying logon passwords in FIX clients and validating logon passwords in FIX servers.
- "raw_data_and_unicode" demonstrates using binary/raw data, unicode characters and sending and handling custom message types.
- "nested_repeating_groups" demonstrates nested repeating groups usage.
- "ssl" demonstrates a FIX server and a FIX client using SSL.
- "tcpdirect_fix_client" demonstrates a FIX client using Solarflare TCPDirect transport.
- "binance_testnet" demonstrates sending custom logon messages, sending and receiving custom FIX messages, and setting header tags.

1.6 Using API

1.6.1 Thread safety

When calling methods from llfix classes, you should ideally do so from a single thread, or ensure that your application logic is implemented in a thread-safe manner.

1.6.2 Engine initialisation and shutdown

llfix::Engine::on_start() must be called at the beginning of your program, and llfix::Engine::shutdown() must be called at the end of your program:

```
#include <llfix/engine.h">
...
...
llfix::Engine::on_start(config_file);
...
instance.shutdown();
llfix::Engine::shutdown();
```

1.6.3 Sending and receiving

Virtual method overrides that handle incoming messages receive an llfix::IncomingFixMessage instance. For non-repeating group tags, you can use the following functions:

- llfix::IncomingFixMessage::has_tag — Checks whether a specific FIX tag exists.
- llfix::IncomingFixMessage::get_tag_value_as — Retrieves the value of a tag as the specified type.

For repeating group tags, use:

- llfix::IncomingFixMessage::has_repeating_group_tag — Checks whether a repeating group tag exists.
- llfix::IncomingFixMessage::get_repeating_group_tag_value_as — Retrieves the value of a tag in a repeating group at the given index.

To send FIX messages:

1. Obtain the outgoing message instance via llfix::FixClient::outgoing_message_instance or llfix::FixServer::outgoing_message_ins
2. Set the message type using llfix::OutgoingFixMessage::set_msg_type .
3. Set the desired tag-value pairs using llfix::OutgoingFixMessage::set_tag . For timestamp tags (e.g., tag 60), use llfix::OutgoingFixMessage::set_timestamp_tag instead.
4. Send the message using llfix::FixClient::send_outgoing_message or llfix::FixServer::send_outgoing_message

1.6.4 Repeating groups

To specify repeating groups for outgoing messages, you can simply use the regular set_tag API:

```
message->set_tag(453, 2);
message->set_tag(448, "PARTY1");
message->set_tag(447, 'D');
message->set_tag(452, 1);
message->set_tag(448, "PARTY2");
message->set_tag(447, 'D');
message->set_tag(452, 3);
```

To decode repeating groups in incoming FIX messages, you can use get_repeating_group_tag_value_as with a zero-based index. Here's an example:

```
if (incoming_message->has_repeating_group_tag(453))
{
    uint32_t repeating_group_no_parties = incoming_message->get_repeating_group_tag_value_as<uint32_t>(453,
       0);
    std::string repeating_group_msg = "Repeating group : ";
    for (std::size_t i = 0; i < repeating_group_no_parties; i++)
    {
        repeating_group_msg += incoming_message->get_repeating_group_tag_value_as<std::string_view>(448, i);
        repeating_group_msg += " ";
        repeating_group_msg += incoming_message->get_repeating_group_tag_value_as<std::string_view>(447, i);
        repeating_group_msg += " ";
        repeating_group_msg += incoming_message->get_repeating_group_tag_value_as<std::string_view>(452, i);
        repeating_group_msg += " ";
    }
}
```

1.6.5 Nested repeating groups

The example below demonstrates encoding the following repeating group :
453=2|448=PARTY1|447=D|452=1|802=2|523=AAA|803=888|523=EEE|803=777|448=PARTY2|447=D|452=3|802=1|523=G↩
GG|803=999|

```
message->set_tag(453, 2);
message->set_tag(448, "PARTY1");
message->set_tag(447, 'D');
message->set_tag(452, 1);
// Nested group1
message->set_tag(802, 2);
message->set_tag(523, "AAA");
message->set_tag(803, "888");
message->set_tag(523, "EEE");
message->set_tag(803, "777");
message->set_tag(448, "PARTY2");
message->set_tag(447, 'D');
message->set_tag(452, 3);
// Nested group2
message->set_tag(802, 1);
message->set_tag(523, "GGG");
message->set_tag(803, "999");
```

The example below demonstrates decoding the same nested repeating group :

```
void log_repeating_group(const llfix::IncomingFixMessage* message)
{
    if (message->has_repeating_group_tag(453))
    {
        uint32_t repeating_group_no_parties = message->get_repeating_group_tag_value_as<uint32_t>(453, 0);
        uint32_t total_nested_group_count = 0;
        std::string repeating_group_msg = "Repeating group : ";
        for (std::size_t i = 0; i < repeating_group_no_parties; i++)
        {
            repeating_group_msg += message->get_repeating_group_tag_value_as<std::string_view>(448, i);
            repeating_group_msg += " ";
            repeating_group_msg += message->get_repeating_group_tag_value_as<std::string_view>(447, i);
            repeating_group_msg += " ";
            repeating_group_msg += message->get_repeating_group_tag_value_as<std::string_view>(452, i);
            repeating_group_msg += " ";
            uint32_t nested_repeating_group_no = message->get_repeating_group_tag_value_as<uint32_t>(802,
       i);
            for(std::size_t j=0; j<nested_repeating_group_no; j++)
```

```
            {
                repeating_group_msg += message->get_repeating_group_tag_value_as<std::string_view>(523,
        total_nested_group_count +j);
                repeating_group_msg += " ";
                repeating_group_msg += message->get_repeating_group_tag_value_as<std::string_view>(803,
        total_nested_group_count +j);
                repeating_group_msg += " ";
            }
            total_nested_group_count += nested_repeating_group_no;
        }
        LLFIX_LOG_INFO(repeating_group_msg);
    }
}
```

You can also find "nested_repeating_groups" example under "tests/other_networked_tests" directory.

1.6.6 Specifying header & trailer tags

If you need static header tags, please refer to the FixSession configuration, where you can use the additional_↩ static_header_tags setting. Static here means that the value will remain unchanged for the entire FIX session.

For non-static header and trailer values, you can override virtal method send_outgoing message and use llfix::OutgoingFixMessage::set_tag as shown below:

```
// Fix client application
bool your_application::send_outgoing_message(llfix::OutgoingFixMessage* message)
{
    // Setting a header tag
    message->set_tag<llfix::FixMessageComponent::HEADER>(tag, value);
    // Setting a trailer tag
    message->set_tag<llfix::FixMessageComponent::TRAILER>(tag, value);
    return llfix::FixClient<Transport>::send_outgoing_message(message);
}
// Fix server application
bool your_application::send_outgoing_message(llfix::FixSession* session, llfix::OutgoingFixMessage* message)
{
    // Setting a header tag
    message->set_tag<llfix::FixMessageComponent::HEADER>(tag, value);
    // Setting a trailer tag
    message->set_tag<llfix::FixMessageComponent::TRAILER>(tag, value);
    return llfix::FixServer<Transport>::send_outgoing_message(session, message);
}
```

You can also check "binance_testnet" example in the "tests/other_networked_tests" directory.

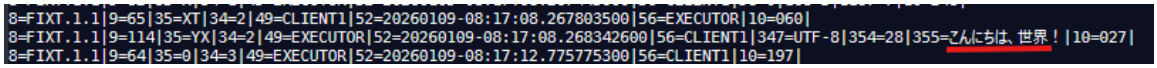
1.6.7 Raw/binary data

To parse FIX messages containing raw (binary) fields, llfix must be built with the following preprocessor definition:

```
-DLLFIX_ENABLE_BINARY_FIELDS
```

When FIX dictionaries are enabled, binary field definitions (e.g. LENGTH/DATA pairs such as t95/t96) are automatically recognised by the FIX client and FIX server parsers.

For a complete example demonstrating the use of binary fields, refer to the "raw_data_and_unicode" example which is located in the tests/other_networked_tests directory. Below is the output of the deserialiser tool for that example :



1.6.8 FixClient overridable virtual methods

1.6.8.1 run()

void llfix::FixClient::run()

Main execution loop of the FIX client.

1.6.8.2 on_connection()

void llfix::FixClient::on_connection()

Called when a connection to the FIX server is established.

1.6.8.3 on_disconnection()

void llfix::FixClient::on_disconnection()

Called when the connection to the FIX server is lost.

1.6.8.4 on_logon_response()

void llfix::FixClient::on_logon_response(const IncomingFixMessage* message)

Called when a successful logon response is received from the server.

1.6.8.5 on_logout_response()

void llfix::FixClient::on_logout_response(const IncomingFixMessage* message)

Called when a logout response is received from the server.

1.6.8.6 on_logon_reject()

void llfix::FixClient::on_logon_reject(const IncomingFixMessage* message)

Called when a logon rejection message is received from the server.

1.6.8.7 on_server_heartbeat()

void llfix::FixClient::on_server_heartbeat()

Called when a heartbeat is received from the server.

1.6.8.8 on_server_test_request()

void llfix::FixClient::on_server_test_request(const IncomingFixMessage* message)

Called when a test request is received from the server.

1.6.8.9 on_server_resend_request()

void llfix::FixClient::on_server_resend_request(const IncomingFixMessage* message)

Called when a resend request is received from the server.

1.6.8.10 on_execution_report()

void llfix::FixClient::on_execution_report(const IncomingFixMessage* message)

Called when an execution report message is received.

1.6.8.11 on_order_cancel_replace_reject()

void llfix::FixClient::on_order_cancel_replace_reject(const IncomingFixMessage* message)

Called when an order cancel/replace reject message is received.

1.6.8.12 on_session_level_reject()

void llfix::FixClient::on_session_level_reject(const IncomingFixMessage* message)

Called when a session-level reject message is received.

1.6.8.13 on_application_level_reject()

void llfix::FixClient::on_application_level_reject(const IncomingFixMessage* message)

Called when an application-level reject message is received.

1.6.8.14 on_custom_message_type()

void llfix::FixClient::on_custom_message_type(const IncomingFixMessage* message)

Called when a custom or user-defined FIX message is received.

1.6.8.15 send_logon_request()

bool llfix::FixClient::send_logon_request()

Handles sending a logon request to the FIX server.

1.6.8.16 send_logout_request()

bool llfix::FixClient::send_logout_request()

Handles sending a logout request to the FIX server.

1.6.8.17 send_client_heartbeat()

bool llfix::FixClient::send_client_heartbeat(FixString* test_request_id)

Handles sending a heartbeat message to the server.

1.6.8.18 send_test_request()

bool llfix::FixClient::send_test_request()

Handles sending a test request message to the server.

1.6.8.19 send_resend_request()

bool llfix::FixClient::send_resend_request()

Handles sending a resend request message to the server.

1.6.8.20 send_sequence_reset_message()

bool llfix::FixClient::send_sequence_reset_message(uint32_t desired_sequence_no)

Handles sending a sequence reset message with the desired sequence number.

1.6.8.21 send_gap_fill_message()

bool llfix::FixClient::send_gap_fill_message()

Handles sending a gap fill sequence reset message.

1.6.8.22 resend_messages_to_server()

void llfix::FixClient::resend_messages_to_server(uint32_t begin_seq_no, uint32_t end_seq_no)

Handles resending messages to the server for the specified sequence range.

send_reject_message()

bool llfix::FixClient::send_reject_message(const IncomingFixMessage& incoming_message, uint32_t reject_↩ reason_code, const char* buffer_message, std::size_t buffer_message_length)

Handles sending a reject message in response to an invalid incoming message.

1.6.8.23 send_outgoing_message()

bool llfix::FixClient::send_outgoing_message(llfix::OutgoingFixMessage* message)

Handles all outgoing messages.

1.6.9 FixServer overridable virtual methods

1.6.9.1 on_new_order()

void llfix::FixServer::on_new_order(FixSession* session, const IncomingFixMessage* message)

Called when a new order message is received from a client.

1.6.9.2 on_cancel_order()

void llfix::FixServer::on_cancel_order(FixSession* session, const IncomingFixMessage* message)

Called when an order cancel request is received from a client.

1.6.9.3 on_replace_order()

void llfix::FixServer::on_replace_order(FixSession* session, const IncomingFixMessage* message)

Called when an order replace request is received from a client.

1.6.9.4 on_application_level_reject()

void llfix::FixServer::on_application_level_reject(FixSession* session, const IncomingFixMessage* message)

Called when an application-level reject is received from a client.

1.6.9.5 on_session_level_reject()

void llfix::FixServer::on_session_level_reject(FixSession* session, const IncomingFixMessage* message)

Called when a session-level reject is received from a client.

1.6.9.6 on_custom_message()

void llfix::FixServer::on_custom_message(FixSession* session, const IncomingFixMessage* message)

Called when a custom or user-defined FIX message is received.

1.6.9.7 on_logon_request()

void llfix::FixServer::on_logon_request(FixSession* session, const IncomingFixMessage* message)

Called when a logon request is received from a client.

1.6.9.8 on_logout_request()

void llfix::FixServer::on_logout_request(FixSession* session, const IncomingFixMessage* message)

Called when a logout request is received from a client.

1.6.9.9 on_client_resend_request()

void llfix::FixServer::on_client_resend_request(FixSession* session, const IncomingFixMessage* message)

Called when a resend request is received from a client.

1.6.9.10 on_client_test_request()

void llfix::FixServer::on_client_test_request(FixSession* session, const IncomingFixMessage* message)

Called when a test request is received from a client.

1.6.9.11 on_client_heartbeat()

void llfix::FixServer::on_client_heartbeat(FixSession* session)

Called when a heartbeat is received from a client.

1.6.9.12 process_incoming_throttling()

bool llfix::FixServer::process_incoming_throttling(FixSession* session, const IncomingFixMessage* incoming_fix_message)

Handles throttling checks for incoming client messages.

1.6.9.13 process_schedule_validator()

void llfix::FixServer::process_schedule_validator(FixSession* session)

Validates the session against configured schedule rules.

1.6.9.14 authenticate_logon_request()

bool llfix::FixServer::authenticate_logon_request(FixSession* session, const IncomingFixMessage* message)

Handles authentication and validation of a logon request.

1.6.9.15 process_logon_request()

void llfix::FixServer::process_logon_request(FixSession* session, std::size_t peer_index, const IncomingFixMessage* message)

Processes an incoming Logon (35=A) request.

1.6.9.16 accept_session()

FixSession* llfix::FixServer::accept_session(std::size_t peer_index, const IncomingFixMessage* incoming_fix_message, const char*

Accepts and validates an incoming session before logon processing.

1.6.9.17 send_heartbeat()

bool llfix::FixServer::send_heartbeat(FixSession* session, FixString* test_request_id)

Handles sending a heartbeat message to the client.

1.6.9.18 send_logon_response()

bool llfix::FixServer::send_logon_response(FixSession* session, const IncomingFixMessage* message)

Handles sending a logon response to the client.

1.6.9.19 send_logout_message()

bool llfix::FixServer::send_logout_message(FixSession* session, const std::string& reason_text = "")

Handles sending a logout message to the client.

1.6.9.20 send_test_request()

bool llfix::FixServer::send_test_request(FixSession* session)

Handles sending a test request to the client.

1.6.9.21 send_resend_request()

bool llfix::FixServer::send_resend_request(FixSession* session)

Handles sending a resend request to the client.

1.6.9.22 send_sequence_reset_message()

bool llfix::FixServer::send_sequence_reset_message(FixSession* session, uint32_t desired_sequence_no)

Handles sending a sequence reset message to the client.

1.6.9.23 send_gap_fill_message()

bool llfix::FixServer::send_gap_fill_message(FixSession* session)

Handles sending a gap fill sequence reset message.

1.6.9.24 resend_messages_to_client()

void llfix::FixServer::resend_messages_to_client(FixSession* session, uint32_t begining_seq_no, uint32_t end_↩ seq_no)

Handles resending messages to the client for the specified sequence range.

1.6.9.25 send_reject_message()

bool llfix::FixServer::send_reject_message(FixSession* session, const IncomingFixMessage* incoming_message, uint32_t reject_reason_code, const char* buffer_message, std::size_t buffer_message_length)

Handles sending a reject message to the client.

1.6.9.26 send_throttle_reject_message()

bool llfix::FixServer::send_throttle_reject_message(FixSession* session, const IncomingFixMessage* incoming↩ _message)

Handles sending a throttle reject message to the client.

1.6.9.27 send_outgoing_message()

bool llfix::FixServer::send_outgoing_message(llfix::FixSession* session, llfix::OutgoingFixMessage* message)

Handles all outgoing messages.

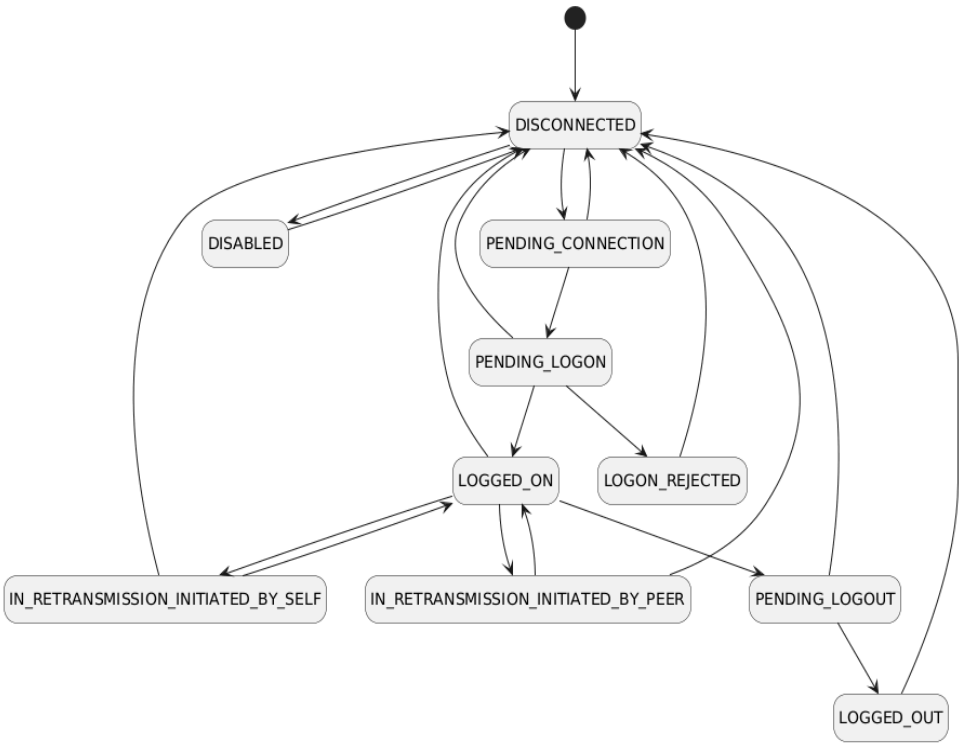
1.6.10 Session states

An llfix engine FIX session can be in one of the following states:

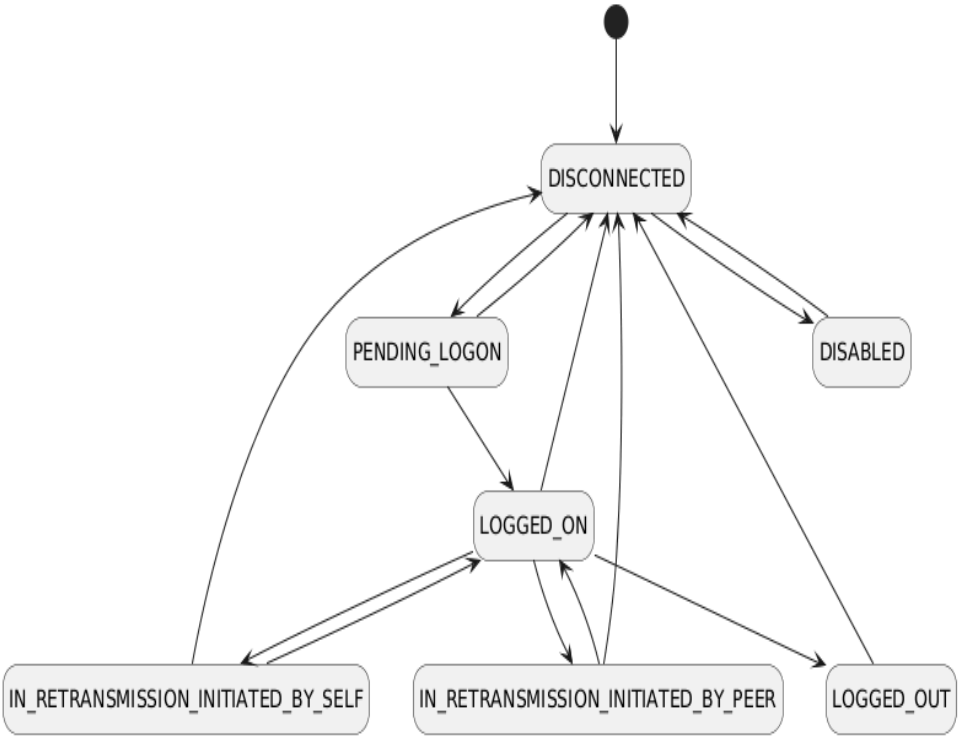
State	Description
DISABLED	Session is administratively disabled.
DISCONNECTED	Session is not connected to the peer.
LOGGED_OUT	Session is disconnected after a successful logout.
LOGON_REJECTED	Logon attempt was rejected by the peer.
PENDING_CONNECTION	TCP connection established, waiting to initiate logon.
PENDING_LOGON	Logon message sent, awaiting logon response.
PENDING_LOGOUT	Logout message sent, awaiting completion.
LOGGED_ON	Session is logged on and fully operational.
IN_RETRANSMISSION_INITIATED_BY_SELF	Session is retransmitting messages initiated by this side.
IN_RETRANSMISSION_INITIATED_BY_PEER	Session is retransmitting messages initiated by the peer.

Note about logouts : On the client, state LOGGED_OUT represents completion of the logout handshake (logout sent and response received). On the server, LOGGED_OUT represents that a Logout message has been received from the peer. The server typically disconnects immediately after sending its Logout response and therefore does not enter a separate PENDING_LOGOUT state.

FIX client state transitions are as below :



FIX server state transitions are as below :



1.6.11 Fixed points

Using fixed points are much more performant compared to floating points and doubles and already most electronic trading venues use fixed-point prices and quantities. llfix engine provides an unsigned fixed point abstraction (llfix::FixedPoint) that can be used as below :

```
// Usage 1
llfix::FixedPoint fp;
fp.set_decimal_points(4);
fp.set_from_chars("1000.0023");
// Actual value is 1000.0023
std::cout « fp.to_string();
// 1000.0023
std::cout « fp.get_raw_value();
// 10000023
// Usage 2
llfix::FixedPoint fp;
fp.set_decimal_points(4);
fp.set_raw_value(10001234);
// Actual value is 1000.1234
std::cout « fp.to_string();
// 1000.1234
std::cout « fp.get_raw_value();
// 10001234
// Encoding in outgoing FIX messages
message->set_tag(TAG_NO, fp);
// Decoding in incoming FIX messages
auto fp = message->get_tag_value_as<llfix::FixedPoint>(TAG_NO);
```

llfix::FixedPoint is a purpose-specific unsigned fixed-point type and is not intended for general-purpose numeric use. Its design imposes several usage constraints; please review the entire llfix::FixedPoint API reference carefully before using it.

1.6.12 TCP Admin interface

When the TCP administration interface is configured, it is started automatically by llfix::Engine::on_start.

Once initialised, you must register your FIX server or FIX client with the management server in order to expose it:

```
...
llfix::Engine::on_start(config_file);
...
// Registering a FIX server after its creation
llfix::Engine::get_management_server().register_server(&server_instance);
// Registering a FIX client after its creation
llfix::Engine::get_management_server().register_client(&client);
...
```

Note that, at the end of your program's lifecycle, the TCP administration interface must be stopped before shutting down your instances:

```
...
llfix::Engine::stop_management_server();
instance.shutdown();
llfix::Engine::shutdown();
```

The examples in the examples directory demonstrate these API calls.

1.6.13 Persistency plugins

You can create your custom message persister by deriving from llfix::MessagePersistPlugin interface and implementing its methods :

```
void persist_incoming_message(const std::string_view& session_name, uint32_t sequence_number, const char*
      buffer, std::size_t buffer_size)
void persist_outgoing_message(const std::string_view& session_name, uint32_t sequence_number, const char*
      buffer, std::size_t buffer_size, bool successfully_transmitted)
```

To enable it you need to call llfix::FixClient::set_message_persist_plugin or llfix::FixServer::set_message_persist_plugin.

You can find "my_message_persister.h" example under "tests/fix_server_automation/server".

When using a custom message persistence plugin, you can disable the built-in message serialisation for a specific session by setting the max_serialised_file_size configuration parameter to 0 for that session.

1.6.14 Session specific data

llfix::FixSession allows applications to associate data with FIX sessions via llfix::FixSession::add_attribute and llfix::FixSession::get_attribute. When adding an attribute, you must specify a name, which should be reused during retrieval. Supported value types:

- std::string
- Arithmetic types (converted via std::to_string)
- Any type supporting stream insertion (operator<<)

An llfix::FixSession can be accessed via llfix::FixClient::get_session or llfix::FixServer::get_session.

1.7 Validations

Note: Session protocol mechanics (sequence number handling, resend requests, gap fills, heartbeats and test requests) are implemented as part of the FIX session state machine and are not listed under validations.

For configuration options to enable or disable validations, please refer to section 1.3.7, FIX Session configs, Validations. Note that fundamental validations, such as parser-level checks, cannot be disabled.

llfix performs validations in several categories:

Parser level

- Missing equals sign (=)
- Tag with no value
- Duplicate tag (excluding repeating groups)
- Non-numeric tag

Message level

- Missing tag 8
- Missing tag 9 or non-numeric tag 9 value
- Missing tag 10 or non-numeric tag 10 value
- Missing tag 34 or non-numeric tag 34 value
- Missing tag 35
- Missing tag 49
- Missing tag 56
- Incorrect body length (tag 9)
- Invalid checksum (tag 10)
- Incorrect tag ordering in headers (tag 8, tag 9, tag 35)

Session level

- Invalid SenderCompID
- Invalid TargetCompID
- Invalid FIX protocol version

Time based

- Tag 52 staleness validation (has its own configuration, disabled by default)

FIX server logons

- Missing tag 98 (encryption method)
- Missing tag 108 (heartbeat interval)
- Invalid tag 108 (heartbeat interval)
- Duplicate logon

Dictionary based

- Invalid message type, message does not exist in the dictionary
- Invalid tag, tag does not exist in the dictionary for a specific message type
- Missing required tag, a required tag for a specific message type is missing
- Undefined tag, field does not exist in the dictionary

Dictionary based, allowed values

Validated data types are as below:

- int
- float
- boolean
- timestamp
- timeonly
- dateonly

Dictionary based, repeating groups

The following validations can be performed for repeating groups that are marked as 'required':

- Missing count/leading tag
- Multiple required group count/leading tags
- Non-numeric count/leading tag

1.8 Administration

1.8.1 Message serialisations & deserialiser tool

Messages are recorded in a binary format for speed per direction (incoming & outgoing). You should use the deserialiser command line tool in order to view the recorded messages.

Deserialiser can be found in "tools/deserialiser" directory. To build it :

- make release on Linux
- run build_msvc.bat or use VisualStudio project file on Windows.

CLI options are :

Option	Description
-i <path>	Input serialisation path
-o <file>	Output file
-e	Exclude timestamps from output
-t	Use tag names instead of numbers

While specifying an input path, it can contain multiple subfolders. It will be recursively searching for all message serialisations and sort them based on serialisation timestamp.

By specifying -t, you can also get textual descriptions instead of tag numbers :

```
...
TIMESTAMP=19:46:28.832665400
Success=1
BeginString=FIX.4.4|BodyLength=188|MsgType=D|MsgSeqNum=2|SenderCompID=CLIENT1|SendingTime=20251229-19:46:28.826044800|TargetC
----------------------------------------
TIMESTAMP=19:46:28.832693500
Success=1
BeginString=FIX.4.4|BodyLength=188|MsgType=D|MsgSeqNum=2|SenderCompID=CLIENT3|SendingTime=20251229-19:46:28.825434700|TargetC
...
```

1.8.2 Sequence stores

Sequence numbers for each session are stored in a binary format. If modification is required for any reason, it can be performed using the Sequence Store Manager tool.

The source code for this tool is located in tools/sequence_store_manager. To build it:

- make release on Linux
- run build_msvc.bat or use VisualStudio project file on Windows.

Once built, the tool can be executed from the command line to modify and verify sequence store files.

1.8.3 Admin Gui

1.8.3.1 Introduction

The llfix engine provides a built-in TCP-based management interface for runtime control, inspection,

This interface is intended for:

- The bundled Python CLI admin client
- The Admin GUI

It is not part of the FIX data plane and does not affect trading latency or session message flow.

The management interface allows operators to:

- Inspect engine runtime information
- Query and manage FIX clients and FIX servers
- Inspect and control FIX sessions
- Perform operational recovery actions (e.g. sequence number control)
- Support high-availability setups (primary / secondary roles)

All operations are performed live, without restarting the engine.

As the interface is intentionally lightweight, it doesn't use encryption. Therefore it must be deployed in a trusted environment.

The Admin GUI is implemented using **Qt 6.7.3**. A list of supported operating systems is available in the Qt 6.7 Supported Platforms documentation.

To run the Admin GUI:

- Windows: The executable and all required DLLs are located in tools/admin_gui/bin_windows. This directory is fully self-contained and can be run directly.
- Ubuntu: Use the admin_gui.sh launcher script found in tools/admin_gui/bin_linux. Ensure the script is executable with "chmod +x ./admin_gui.sh". This directory includes all required shared libraries and is fully self-contained.

1.8.3.2 Building

The Admin GUI is implemented using Qt 6.7.3 and dynamically links against the Qt shared libraries (shared objects on Linux and DLLs on Windows).

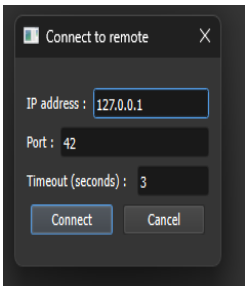
To build the Admin GUI, Qt Creator must be installed on the system.

Once Qt Creator is available, open the provided Qt project file admin_gui.pro, located in tools/admin_gui/src, and build the project using Qt Creator.

1.8.3.3 Usage

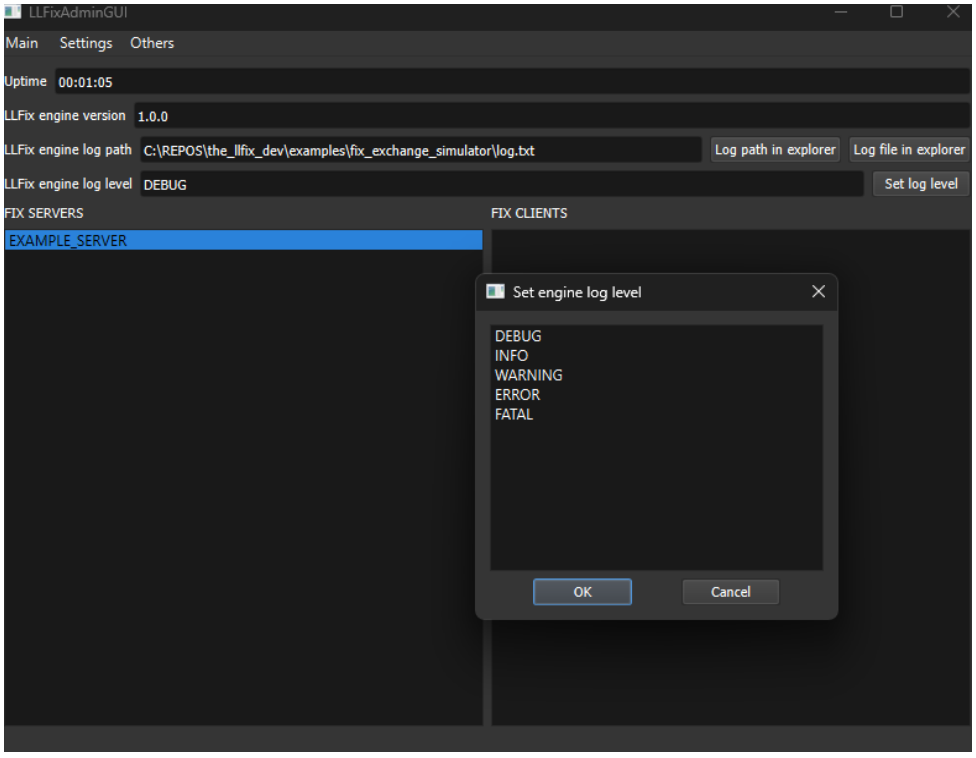
To connect to an instance running the llfix engine:

1. From the Main menu, select Connect.
2. In the connection dialog, specify the IP address and port, then click Connect



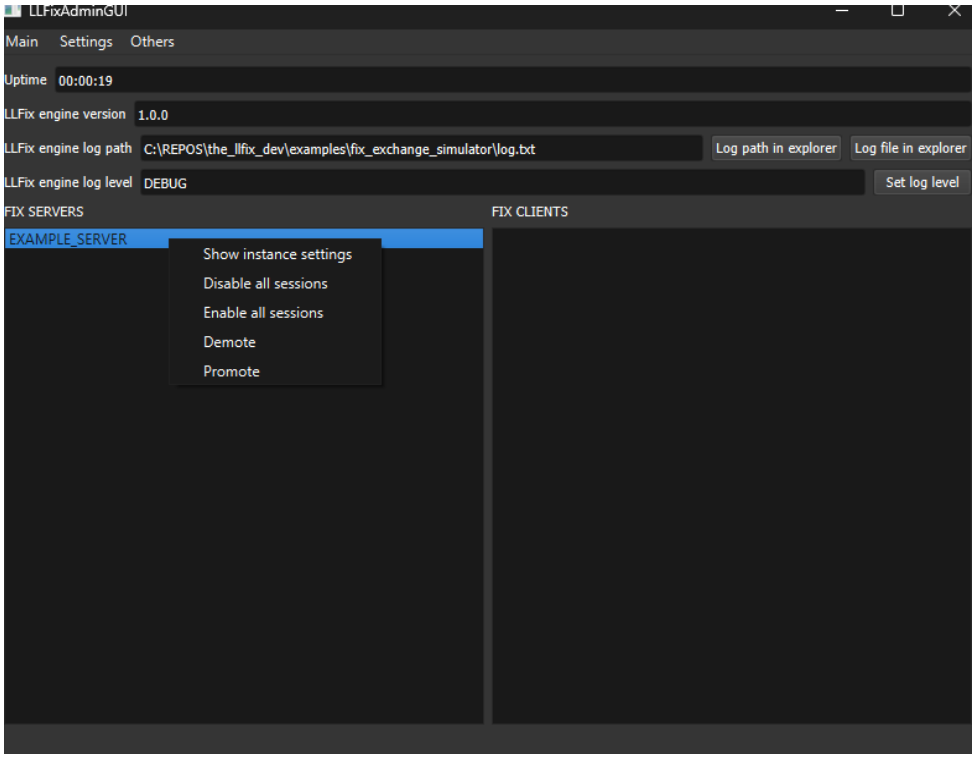
After establishing a connection, you will be taken to the main screen. This screen displays the uptime, llfix engine version, current log level, and all registered FIX server and FIX client instances.

From this screen, you can change the log level without restarting the running application :



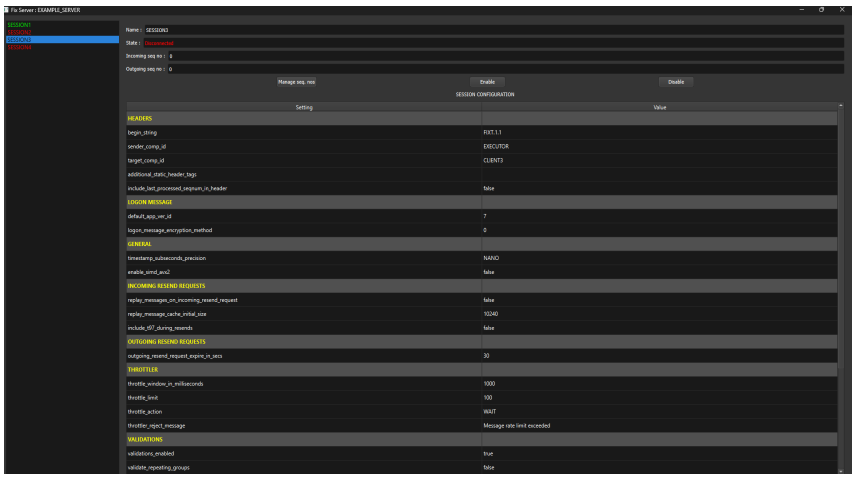
If you right-click on any instance, the instance context menu provides the following options:

- Show Loaded Instance Configuration
- Disable All Sessions
- Enable All Sessions
- Demote (High Availability)
- Promote (High Availability)



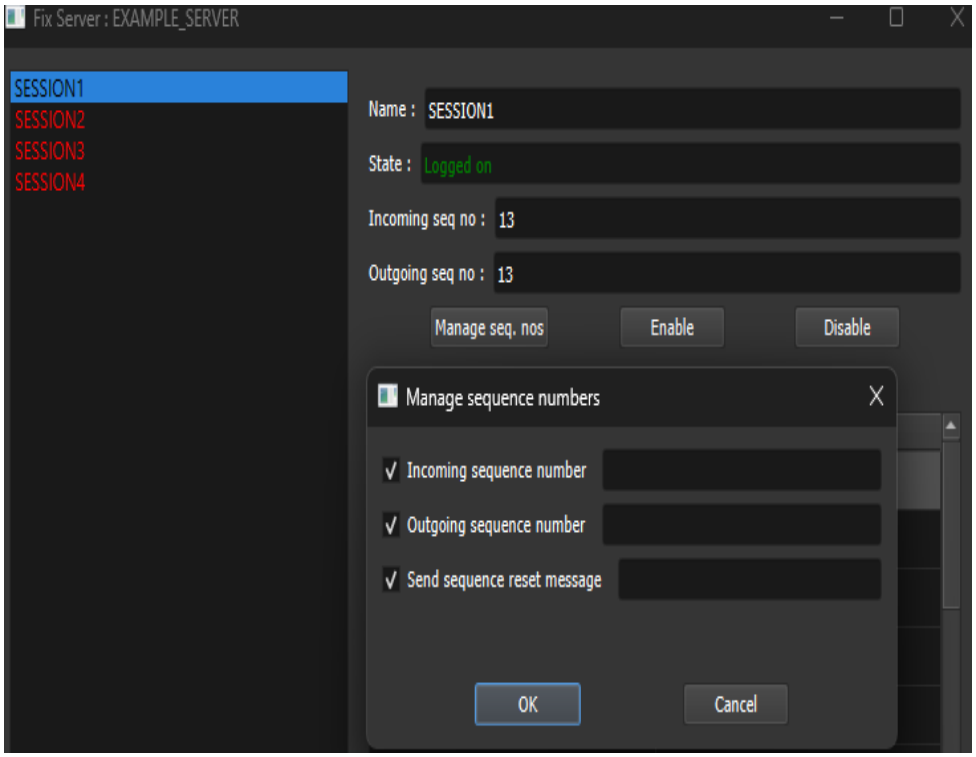
After double-clicking an instance name, the Instance Dialog opens, displaying a list of sessions within that instance. Clicking on a session name will open the Session Screen.

In the Session Screen, you can view the session state, sequence numbers, and the loaded configuration information :

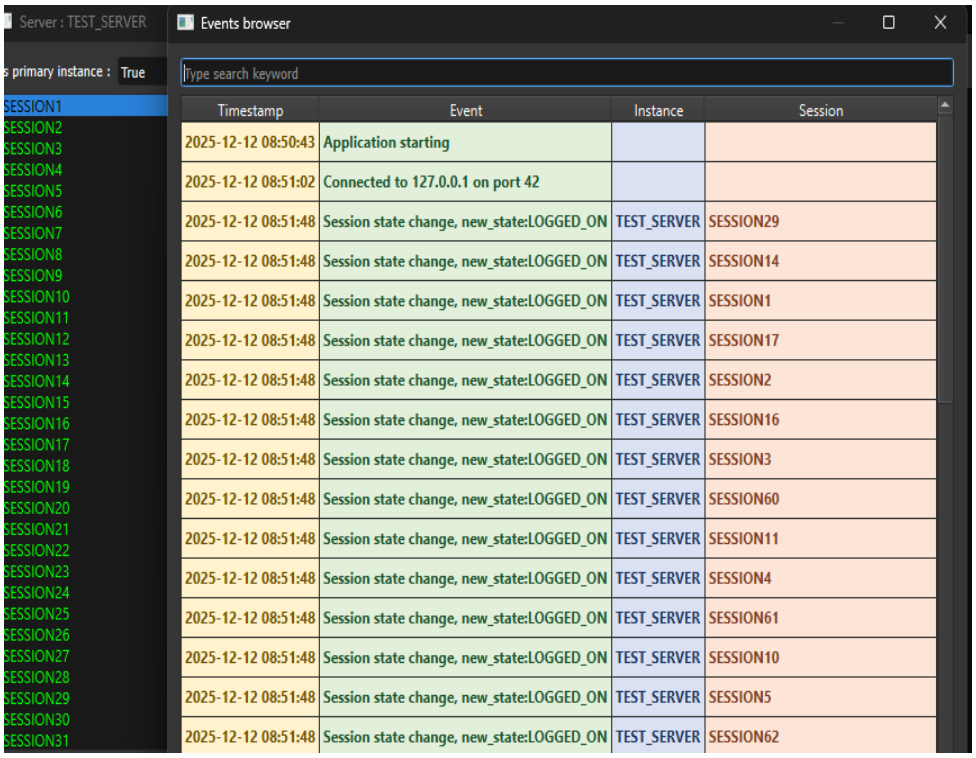


Clicking “Manage Seq. Nos” opens the Session Sequence Number Management screen. From here, you can:

- Set the incoming sequence number
- Set the outgoing sequence number
- Send a Sequence Reset message to the connected peer



From the Main Menu, click “Others” and then “Events Browser” to open the Events Browser window. Here, you can monitor and filter events such as session state changes, sent administrative commands, and more.



1.8.3.4 Port forwarding

To remain lightweight, the Ilfix engine and Admin GUI communicate over plain TCP. In environments where encrypted transport is required, such as accessing a colocation site over SSH, TCP traffic can be secured using SSH port forwarding.

The steps below describe how to configure local port forwarding using PuTTY:

1. Launch PuTTY
2. Navigate to Connection → SSH → Tunnels.
3. In the Source port field, enter the local port on your client machine.
4. In the Destination field, specify the IP address and port of the remote TCP server.
5. Ensure the Local radio button is selected.
6. Click Add to register the forwarding rule.
7. Return to the Session category and click Open to initiate the SSH connection.
8. After logging in to the remote server, the port forwarding tunnel is active.

1.8.4 Admin CLI

You can find "admin_client.py" under tools directory. Its usage is as below:

```
python admin_client.py <server> <port_number>
```

The table below lists the available admin interface commands and their descriptions :

Command	Description
help	Display all available admin CLI commands
get_uptime	Show engine uptime since startup
get_engine_version	Return the running Ilfix engine version
get_engine_log_path	Display the current engine log file path
get_engine_log_level	Show the active engine log level
set_engine_log_level <log_level>	Set engine log level (ERROR, WARNING, INFO, D↩ EBUG)
get_clients	List all registered FIX client instances
get_servers	List all registered FIX server instances
get_instance_config <client/server>	Display configuration of a client or server instance
set_is_instance_primary <c/s> <1/0>	Promote (1) or demote (0) an instance to primary role
is_instance_primary <client/server>	Check whether an instance is currently primary
get_sessions <client/server>	List all FIX sessions for a given instance
get_session_state <c/s> <session>	Return the current state of a FIX session
get_session_config <c/s> <session>	Display configuration of a FIX session
get_all_session_states	Show states of all sessions across all instances
enable_session <c/s> <session>	Enable a FIX session
disable_session <c/s> <session>	Disable a FIX session
get_incoming_sequence_number <c/s> <session>	Get current incoming sequence number
get_outgoing_sequence_number <c/s> <session>	Get current outgoing sequence number
set_incoming_sequence_number <c/s> <session> <no>	Set incoming sequence number
set_outgoing_sequence_number <c/s> <session> <no>	Set outgoing sequence number
send_sequence_reset <c/s> <session> <no>	Send FIX sequence reset

1.8.5 Logs

You can find the logger configuration options in the Engine Configs section of the Ilfix documentation.

The Ilfix logger supports the following severity levels:

FATAL, ERROR, WARNING, INFO, DEBUG

During initialisation, log messages use the INFO, WARNING, and ERROR severities. The FATAL severity is reserved for critical failures, such as memory allocation errors or failures when creating memory-mapped files.

Validation errors are logged using the ERROR severity.

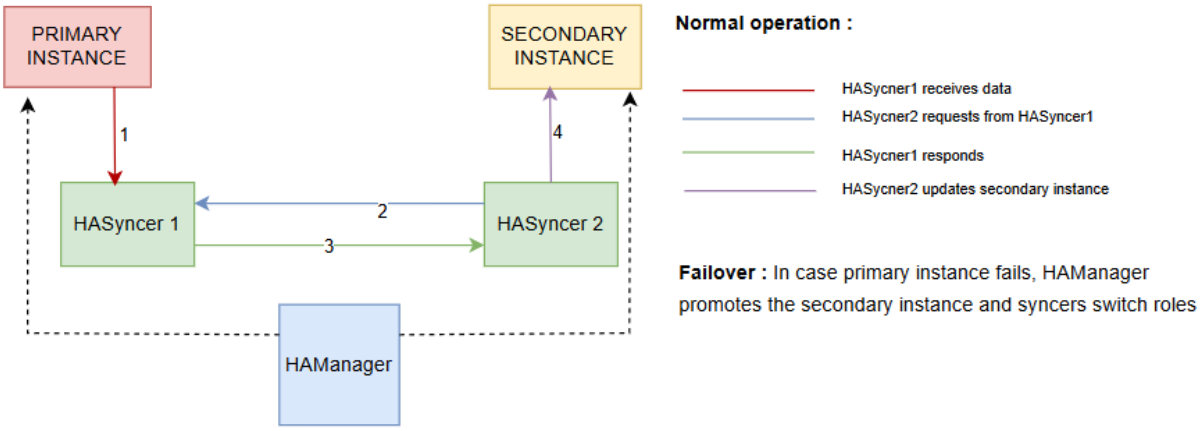
Note that the logger severity can be changed at runtime via the Admin GUI without restarting the instance.

1.9 High availability

1.9.1 Overview

Sequence numbers and messages are synchronised reliably using a TCP-based HA Syncer. Synchronisation follows a receiver-driven model to minimise network load.

Automatic promotion and demotion ensure uninterrupted FIX sessions, while the HA Manager guarantees continuous availability through TCP-based leader election. Manual failovers can also be performed via the Admin GUI.



1.9.2 Primary and Secondary Instance Roles

Each FIX server/client instance operates in either a primary or secondary role, as determined by the starts_as_↩ primary_instance configuration setting.

- When starts_as_primary_instance is set to false, the instance starts in the secondary role and all FIX sessions are disabled.
- When starts_as_primary_instance is set to true, the instance starts in the primary role and all FIX sessions are enabled.

1.9.3 HA Syncer

The HA Syncer executable operates in dual mode, meaning it can both send and receive data. The initially_↩ supporting_primary_instance configuration determines whether the parent instance starts as primary or secondary :

1. If the parent instance is secondary: The Syncer operates as a receiver. It actively sends synchronisation queries
2. If the parent instance is primary: The Syncer operates as a sender. It responds to incoming synchronisation queries

Each syncer runs an admin client connected to its parent instance to query the current HA role (primary or secondary).

If the parent instance's admin interface is unreachable, the Syncer enters an unknown state and remains inactive until connectivity is restored.

Data transfer is receiver-driven, as outlined below:

- The secondary instance's Syncer (receiver) sends a query for a specific session.
- The primary instance's Syncer (sender) responds with the sequence numbers and the latest messages.
- After receiving a response, the receiver updates sequence numbers and sends requests for new messages.
- A configurable timeout is available so that if no response is received within the specified time, the receiver resends another query.

1.9.4 Promotions and demotions

Both FIX client and FIX server instances support the administrative command set_is_instance_primary, which controls whether an instance operates as a primary or secondary node :

- When an instance receives set_is_instance_primary 0, all FIX sessions are disabled.
- When an instance receives set_is_instance_primary 1, all FIX sessions are enabled.

Syncers determine the operational state of their parent instance by issuing the is_instance_primary query. Based on the returned value, a syncer will automatically switch its mode between receiver and sender.

1.9.5 HA Manager and automatic failovers

The HA Manager executable is responsible for high-availability orchestration:

- It detects failure of the current primary instance and promotes a secondary instance to primary.
- It also enforces correctness by demoting any instance that incorrectly claims to be primary, including a previously failed primary or any other rogue instance.

Note that instance promotion and demotion can also be performed manually via the administrative GUI.

1.9.6 Setup

You can find an example setup under "tests/other_networked_tests/high_availability" folder.

1.9.6.1 FIX Server/Client instances

1. Both primary and secondary instances should be running TCP admin interfaces. Please refer to engine configs.
2. Primary instance should have starts_as_primary_instance=true config.
3. Secondary instance should have starts_as_primary_instance=false config.

1.9.6.2 HA Syncers

HASyncer is located in the tools/high_availability_syncer directory. To build it:

- On Linux: run make release
- On Windows: run build_msvc.bat or open the Visual Studio project file

You will need two configuration files: one for the primary instance and one for the secondary instance. Example configs are provided in the tools/high_availability_syncer/ directory.

HASyncer configuration descriptions:

Config	Description	Default
initially_supporting_primary_instance	Whether the parent instance initially acts as the primary	false
log_file	Path to the file where logs will be written	log.txt
log_level	Logging verbosity level. One of : ERROR WARNING INFO DEBUG	INFO
log_queries	Enable or disable logging of query messages	false
log_message_syncs	Enable or disable logging of message sync messages	false
sender_nic_ip	IP address of the network interface card used by the sender	
sender_port	Port number the sender will use for communication	
sender_cpu_core_id	CPU core ID to which the sender thread is pinned. -1 disables pinning	-1
receiver_nic_ip	IP address of the network interface card used by the receiver	
receiver_target_ip	Target IP address the receiver will listen to	
receiver_target_port	Target port number the receiver will listen to	
receiver_cpu_core_id	CPU core ID to which the receiver thread is pinned. -1 disables pinning	-1
sync_query_timeout_microseconds	Timeout in microseconds for sync queries	100000
sync_outgoing_messages	Synchronise outgoing FIX messages	true
sync_incoming_messages	Synchronise incoming FIX messages	true
max_request_message_count_per_query_response	Maximum number of incoming or outgoing messages that can be requested per query	100
parent_instance_name	Name of the parent instance to connect to	
parent_instance_admin_ip	IP address of the parent instance's admin interface	
parent_instance_admin_port	Port number of the parent instance's admin interface	
parent_instance_admin_client_nic_ip	NIC IP used by the admin client when connecting to parent	
parent_instance_admin_client_query_interval_milliseconds	Interval in milliseconds between admin client queries	1000

Additionally, an HASyncer configuration must include settings for the sessions it will be syncing. The example below defines four different sessions. Note that all session attributes must begin with SESSION:

```
[SESSION_ONE]
sequence_store_file_path=./test_suite/server1/clients/client1/sequence.store
max_serialised_file_size=67108864
incoming_message_serialisation_path=./test_suite/server1/clients/client1/messages_incoming
outgoing_message_serialisation_path=./test_suite/server1/clients/client1/messages_outgoing
[SESSION_TWO]
sequence_store_file_path=./test_suite/server1/clients/client2/sequence.store
max_serialised_file_size=67108864
incoming_message_serialisation_path=./test_suite/server1/clients/client2/messages_incoming
outgoing_message_serialisation_path=./test_suite/server1/clients/client2/messages_outgoing
[SESSION_THREE]
sequence_store_file_path=./test_suite/server1/clients/client3/sequence.store
max_serialised_file_size=67108864
incoming_message_serialisation_path=./test_suite/server1/clients/client3/messages_incoming
outgoing_message_serialisation_path=./test_suite/server1/clients/client3/messages_outgoing
[SESSION_FOUR]
sequence_store_file_path=./test_suite/server1/clients/client4/sequence.store
max_serialised_file_size=67108864
incoming_message_serialisation_path=./test_suite/server1/clients/client4/messages_incoming
outgoing_message_serialisation_path=./test_suite/server1/clients/client4/messages_outgoing
```

Important: Configured sequence stores must be available before starting HASyncer. In other words, the FIX client/server instances should be started first.

1.9.6.3 HA Manager

HAManager is located in the tools/high_availability_manager directory. To build it:

- make release on Linux
- run build_msvc.bat or use VisualStudio project file on Windows.

You will need one configuration file for your single HAManager instance. An example config is provided in the tools/high_availability_manager/ directory.

HAManager configuration descriptions:

Config	Description	Default
log_file	Path to the file where logs will be written	log.txt
log_level	Logging verbosity level. One of : ERROR WARNING INFO DEBUG	INFO
failover_timeout_milliseconds	Maximum time to wait for a failover to complete before giving up	10
promotion_try_wait_milliseconds	Delay between attempts to promote a secondary instance to primary	1
demotion_try_wait_milliseconds	Delay between attempts to demote a misbehaving or rogue primary	1
initial_leader_finding_timeout_milliseconds	Timeout for discovering the current primary when starting up	1000
initial_new_leader_assignment_timeout_milliseconds	Timeout for assigning a new leader during initial startup	2000

Additionally, an HAManager configuration must include settings for instances it will be managing. The example below defines two instances. Note that all instance attributes must begin with INSTANCE:

```
[INSTANCE_ONE]
instance_name=SERVER1
instance_admin_ip=192.168.10.103
instance_admin_port=42
instance_admin_client_nic_ip=192.168.10.103
instance_admin_client_query_interval_milliseconds=1000
[INSTANCE_TWO]
instance_name=SERVER2
instance_admin_ip=192.168.10.100
instance_admin_port=44
instance_admin_client_nic_ip=192.168.10.100
instance_admin_client_query_interval_milliseconds=1000
```

1.10 Working with Solarflare TCPDirect

1.10.1 Overview

Solarflare Onload accelerates networking by implementing the BSD sockets API in user space, allowing applications to run without recompilation. Solarflare TCPDirect bypasses the BSD sockets interface and its associated semantics to achieve even lower latency.

In llfix, you must derive your FIX client implementation from llfix::FixClient<llfix::TCPConnectorTCPDirect> in order to use TCPDirect for your FIX clients. Additionally you also must specify nic_name and nic_address configurations for your FIX clients.

You can also find "tcpdirect_fix_client" example under "tests/other_networked_tests" directory.

There are several important differences compared with regular operating system sockets:

- Solarflare TCPDirect does not support loopback. To operate correctly, the NIC must have an active physical link.
- When TCPDirect is used, the data path differs from the standard kernel networking stack. As a result, conventional packet capture tools will not function. For troubleshooting, you can use the zf_stackdump utility, which is documented in the the official Solarflare TCPDirect documentation. For packet capture capabilities, please contact AMD regarding their toolchain.

1.10.2 Huge pages requirement

TCPDirect relies on virtual memory huge pages. As a result, the system must be configured with a sufficient number of huge pages before TCPDirect can be used.

TCPDirect utilises virtual memory huge pages; therefore, your system must be configured with a sufficient number of huge pages. Each TCPDirect stack typically consumes at least 10 huge pages, and the llfix implementation uses one TCPDirect stack per FixClient.

If you want to reduce huge page consumption, see **Using shared TCPDirect Stacks** below.

1.10.3 Limited PIO resources

PIO resources are limited on Solarflare NICs. This limitation constrains the maximum number of FIX clients that can use private TCPDirect stacks concurrently.

Once available PIO resources are exhausted, additional TCPDirect stacks cannot be created. For more details, refer to the official Solarflare TCPDirect documentation.

To mitigate PIO resource limitations, see **Using shared TCPDirect Stacks** below.

1.10.4 Using shared TCPDirect stacks

It is possible to reuse a **shared TCPDirect stack** across multiple FIX client instances in order to:

- Reduce huge page consumption
- Increase the number of FIX clients

The "tcpdirect_fix_client" example under "tests/other_networked_tests" also demonstrates how to create and use shared TCPDirect stacks.

IMPORTANT: When a TCPDirect stack is shared, it must be accessed by only a single thread. Unlike private stacks, shared stacks do not support multi-threaded FIX clients. Consequently, all FIX clients using a shared TCPDirect stack must operate in a single-threaded manner.

1.10.5 Continuous incoming data processing requirement

Compared with operating system sockets, TCPDirect provides greater control but requires more manual management. When writing your application, you must ensure that llfix::FixClient::process is called sufficiently often, as this call continuously advances the TCPDirect stack.

It is important to note that this requirement applies even to applications that only transmit data and do not explicitly receive any. In such cases, llfix::FixClient::process must still be invoked in order to process TCP acknowledgements for the TCPDirect stack. Failing to do so may lead to jitter. high tail latencies and hangs.

1.11 SSL

1.11.1 Overview

The outline of the SSL implementation is as follows:

- The supported OpenSSL library version is 3.6 or later, with 3.6 as the minimum requirement.
- The supported certificate and key file format is PEM. You can use OpenSSL utilities to convert certificates from other formats, such as PKCS#12 (.p12 / .pfx), to PEM if required.

1.11.2 Sigpipe signal

The current implementation configures the entire process to ignore SIGPIPE signals on Linux. This is primarily due to the fact that SSL_write does not provide an equivalent to the MSG_NOSIGNAL option available with BSD sockets.

1.11.3 Continuous incoming data processing requirement

When using OpenSSL, you must ensure that llfix::FixClient::process() is called frequently enough to adequately drain the receive (RX) path. Failing to do so can prevent the TLS state machine from making progress and may lead to stalls, hangs, or apparent deadlocks during send operations.

1.12 Tuning

1.12.1 System

For optimal server operation, the following conditions are recommended:

- CPU frequency should be maximised to ensure consistent low-latency performance.
- A sufficient number of isolated CPU cores should be available. Core isolation and pinning are supported by llfix through the relevant CPU core ID configuration parameters.

For further guidance on system-level tuning, the RHEL Low Latency Tuning Guide is a useful reference, as it provides comprehensive recommendations for optimising operating system and network performance.

1.12.2 llfix settings

Several llfix configuration parameters may affect performance:

- FixClient, FixServer, and the TCP administration interface expose CPU core ID configuration options, which are used for thread pinning. Pinning threads to specific cores reduces context switching and improves cache locality, resulting in lower latency and more predictable performance.
- Setting enable_simd_avx2=true enables SIMD acceleration for FIX checksum calculation and validation.
- The numa_bind_node and numa_aware_allocations may be used to influence the initial memory allocations performed by llfix. Note that for numa_aware_allocations to take effect, the numa_bind_node parameter must also be specified. NUMA-aware allocation can reduce memory access latency.

1.12.3 llfix::FixedPoint

The use of llfix::FixedPoint is strongly recommended in preference to floating-point types (float and double) wherever possible. The llfix::FixedPoint class is described in detail in the Using the API section.

1.12.4 Dictionary trimming

Remove all unused FIX fields, message types, and repeating groups. A leaner dictionary reduces validation overhead and results in more predictable latency.

1.13 Third party licences

The product uses or optionally supports the following third-party components, along with their respective licences:

- Deserialiser and code generator tools use cxxopts library (MIT) : https://github.com/jarro2783/cxxopts
- AdminGUI uses Qt (GNU LGPL) : https://www.qt.io/licensing
- Benchmarks and test suites use Quickfix and Quickfix dictionaries (The QuickFIX Software Licence) : https://github.com/quickfix/quickfix
- Benchmarks use Fix8 (LGPL-3.0) : https://github.com/fix8/fix8
- Test suites use tinyfix (MIT) : https://github.com/CorewareLtd/tinyfix
- Optional AMD Solarflare TCPDirect (MIT) : https://www.amd.com/en/support/ethernet-adapters/solarf
  html#downloads
- Optional LibNUMA (GNU LGPL) : https://github.com/numactl/numactl
- Optional tinyxml2 (Zlib) : https://github.com/leethomason/tinyxml2
- Optional OpenSSL (Apache 2.0) : https://github.com/openssl/openssl
- Documentation is generated using Doxygen (GPL v2) : https://www.doxygen.nl/
- Documentation styling uses doxygen-dark-theme (MIT): https://github.com/MaJerle/doxygen-dark-theme

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

Class Documentation

3.1 llfix::Engine Class Reference

Singleton Engine instance.

`#include <engine.h>`

Static Public Member Functions

- static void on_start (const std::string &config_file_path="", const std::string &config_group_name="ENGINE")

  Initialise and start the engine.
- static void shutdown ()

  Stops the engine by releasing resources.
- static void stop_management_server ()

  Stops the management server.
- static ManagementServer & get_management_server ()

  Get reference to the management server instance.

3.1.1 Detailed Description

Singleton Engine instance.

This class is responsible for managing subsystems that are shared between FIX instances : TCP admin interface, logging, allocations, thread local storage, OpenSSL and Solarflare TCPDirect

3.1.2 Member Function Documentation

3.1.2.1 get_management_server()

```
static ManagementServer& llfix::Engine::get_management_server ( )  [inline], [static]
```

Get reference to the management server instance.

Returns

Reference to the singleton ManagementServer instance.

3.1.2.2 on_start()

```
static void llfix::Engine::on_start (
            const std::string & config_file_path = "",
            const std::string & config_group_name = "ENGINE" )  [inline], [static]
```

Initialise and start the engine.

Should be called before initialising any FIX client/FIX server Note : Not thread safe

Parameters

config_file_path	Optional path to configuration file.
config_group_name	Optional configuration group name (default "ENGINE").

3.1.2.3 shutdown()

```
static void llfix::Engine::shutdown ( )  [inline], [static]
```

Stops the engine by releasing resources.

Safe to call multiple times. Ensures the management server is stopped cleanly.

3.1.2.4 stop_management_server()

```
static void llfix::Engine::stop_management_server ( )  [inline], [static]
```

Stops the management server.

Safe to call multiple times. Ensures the management server is stopped cleanly.

The documentation for this class was generated from the following file:

- include/llfix/engine.h

3.2 llfix::FixClient< Transport > Class Template Reference

FIX client implementation.

`#include <fix_client.h>`

Inherits Transport, and ManagedInstance.

Public Member Functions

- bool create (const std::string &client_name, const FixClientSettings &settings, const std::string &session_↩ name, const FixSessionSettings &session_settings)

  Create and initialise a FIX client instance.
- bool create (const std::string &client_config_file_path, const std::string &client_name, const std::string &session_config_file_path, const std::string &session_name)

  Create and initialise a FIX client using configuration files.
- FixSession * get_session (const std::string &session_name="") override

  Retrieve the FIX session managed by this client.
- template<typename... Args>
  void specify_repeating_group (Args... args)

  Specify repeating group definitions for incoming FIX messages.
- bool start ()

  Start the client execution thread.
- bool connect ()

  Establish a FIX network connection and initiate logon.
- void process ()

  Process incoming and outgoing FIX protocol activity.
- void shutdown (bool graceful_shutdown=true)

  Shutdown the FIX client.
- OutgoingFixMessage * outgoing_message_instance ()

  Obtain a reusable outgoing FIX message instance.
- virtual bool send_outgoing_message (OutgoingFixMessage *message)

  Encode and send an outgoing FIX message.
- bool connected_to_primary () const

  Check if the client is connected to the primary endpoint.
- bool connected_to_secondary () const

  Check if the client is connected to the secondary endpoint.
- virtual void on_connection ()

  Called when a TCP connection is successfully established.
- virtual void on_disconnection ()

  Called when the TCP connection is lost or closed.
- virtual void on_logon_response (const IncomingFixMessage *message)

  Called upon receiving a successful FIX Logon response.
- virtual void on_logout_response (const IncomingFixMessage *message)

  Called upon receiving a FIX Logout response.
- virtual void on_logon_reject (const IncomingFixMessage *message)

  Called when a FIX Logon is rejected.
- virtual void on_server_heartbeat ()

  Called upon receiving a FIX Heartbeat from the server.
- virtual void on_server_test_request (const IncomingFixMessage *message)

Called upon receiving a FIX Test Request.

- virtual void on_server_resend_request (const IncomingFixMessage *message)

  Called upon receiving a FIX Resend Request.
- virtual void on_execution_report (const IncomingFixMessage *message)

  Called when an Execution Report (35=8) is received.
- virtual void on_order_cancel_replace_reject (const IncomingFixMessage *message)

  Called when an Order Cancel/Replace Reject (35=9) is received.
- virtual void on_session_level_reject (const IncomingFixMessage *message)

  Called on session-level FIX rejects.
- virtual void on_application_level_reject (const IncomingFixMessage *message)

  Called on application-level FIX rejects.
- virtual void on_custom_message_type (const IncomingFixMessage *message)

  Called for other FIX message types.
- void set_message_persist_plugin (MessagePersistPlugin *plugin)

  Set the message persistence plugin.

Protected Member Functions

- virtual bool send_logon_request ()

  Send FIX Logon request.
- virtual bool send_logout_request ()

  Send FIX Logout request.
- virtual bool send_client_heartbeat (FixString *test_request_id)

  Send FIX Heartbeat message.
- virtual bool send_test_request ()

  Send FIX Test Request message.
- virtual bool send_resend_request ()

  Send FIX Resend Request message.
- virtual bool send_sequence_reset_message (uint32_t desired_sequence_no)

  Send FIX Sequence Reset message.
- virtual bool send_gap_fill_message ()

  Send FIX Gap Fill message.

3.2.1 Detailed Description

template<typename Transport>
class llfix::FixClient< Transport >

FIX client implementation.

Template Parameters

Transport	Transport layer implementation: one of TCPConnector TCPConnectorTCPDirect TCPConnectorSSL.

FixClient manages a single FIX session lifecycle including connection management, logon/logout, message processing, sequencing, retransmissions, and heartbeats.

The class can run in a dedicated internal thread or be driven externally via repeated calls to process().

Users are expected to derive from FixClient and override virtual callbacks to handle session and application-level FIX messages.

3.2.2 Member Function Documentation

3.2.2.1 connect()

```
template<typename Transport >
bool llfix::FixClient< Transport >::connect ( )  [inline]
```

Establish a FIX network connection and initiate logon.

Returns

true if connection and logon initiation succeeded, false otherwise.

Attempts connection to the primary endpoint first and falls back to the secondary endpoint if configured.

3.2.2.2 connected_to_primary()

```
template<typename Transport >
bool llfix::FixClient< Transport >::connected_to_primary ( ) const  [inline]
```

Check if the client is connected to the primary endpoint.

Returns

true if connected to primary, false otherwise.

3.2.2.3 connected_to_secondary()

```
template<typename Transport >
bool llfix::FixClient< Transport >::connected_to_secondary ( ) const  [inline]
```

Check if the client is connected to the secondary endpoint.

Returns

true if connected to secondary, false otherwise.

3.2.2.4 create() [1/2]

```
template<typename Transport >
bool llfix::FixClient< Transport >::create (
            const std::string & client_config_file_path,
            const std::string & client_name,
            const std::string & session_config_file_path,
            const std::string & session_name )  [inline]
```

Create and initialise a FIX client using configuration files.

Parameters

client_config_file_path	Path to client configuration file.
client_name	Client name inside the config.
session_config_file_path	Path to session configuration file.
session_name	Session name inside the config.

Returns

true on success, false if configuration loading or validation fails.

3.2.2.5 create() [2/2]

```
template<typename Transport >
bool llfix::FixClient< Transport >::create (
            const std::string & client_name,
            const FixClientSettings & settings,
            const std::string & session_name,
            const FixSessionSettings & session_settings )  [inline]
```

Create and initialise a FIX client instance.

Parameters

client_name	Logical name of the FIX client.
settings	Client-level configuration.
session_name	Name of the FIX session.
session_settings	Session-level configuration.

Returns

true on successful creation, false otherwise.

3.2.2.6 get_session()

```
template<typename Transport >
FixSession* llfix::FixClient< Transport >::get_session (
            const std::string & session_name = "" )  [inline], [override]
```

Retrieve the FIX session managed by this client.

Returns

Pointer to the FixSession instance.

3.2.2.7 on_application_level_reject()

```
template<typename Transport >
virtual void llfix::FixClient< Transport >::on_application_level_reject (
            const IncomingFixMessage * message )  [inline], [virtual]
```

Called on application-level FIX rejects.

Parameters

message	Incoming reject message.

3.2.2.8 on_custom_message_type()

```
template<typename Transport >
virtual void llfix::FixClient< Transport >::on_custom_message_type (
            const IncomingFixMessage * message )  [inline], [virtual]
```

Called for other FIX message types.

Parameters

message	Incoming FIX message.

3.2.2.9 on_execution_report()

```
template<typename Transport >
virtual void llfix::FixClient< Transport >::on_execution_report (
            const IncomingFixMessage * message )  [inline], [virtual]
```

Called when an Execution Report (35=8) is received.

Parameters

message	Incoming execution report message.

3.2.2.10 on_logon_reject()

```
template<typename Transport >
virtual void llfix::FixClient< Transport >::on_logon_reject (
            const IncomingFixMessage * message )  [inline], [virtual]
```

Called when a FIX Logon is rejected.

Parameters

message	Incoming FIX reject message.

3.2.2.11 on_logon_response()

```
template<typename Transport >
virtual void llfix::FixClient< Transport >::on_logon_response (
            const IncomingFixMessage * message )  [inline], [virtual]
```

Called upon receiving a successful FIX Logon response.

Parameters

message	Incoming FIX logon message.

3.2.2.12 on_logout_response()

```
template<typename Transport >
virtual void llfix::FixClient< Transport >::on_logout_response (
            const IncomingFixMessage * message )  [inline], [virtual]
```

Called upon receiving a FIX Logout response.

Parameters

message	Incoming FIX logout message.

3.2.2.13 on_order_cancel_replace_reject()

```
template<typename Transport >
virtual void llfix::FixClient< Transport >::on_order_cancel_replace_reject (
            const IncomingFixMessage * message )  [inline], [virtual]
```

Called when an Order Cancel/Replace Reject (35=9) is received.

Parameters

message	Incoming reject message.

3.2.2.14 on_server_resend_request()

```
template<typename Transport >
virtual void llfix::FixClient< Transport >::on_server_resend_request (
            const IncomingFixMessage * message )  [inline], [virtual]
```

Called upon receiving a FIX Resend Request.

Parameters

message	Incoming FIX resend request message.

3.2.2.15 on_server_test_request()

```
template<typename Transport >
virtual void llfix::FixClient< Transport >::on_server_test_request (
            const IncomingFixMessage * message )  [inline], [virtual]
```

Called upon receiving a FIX Test Request.

Parameters

message	Incoming FIX test request message.

3.2.2.16 on_session_level_reject()

```
template<typename Transport >
virtual void llfix::FixClient< Transport >::on_session_level_reject (
            const IncomingFixMessage * message )  [inline], [virtual]
```

Called on session-level FIX rejects.

Parameters

message	Incoming reject message.

3.2.2.17 outgoing_message_instance()

```
template<typename Transport >
OutgoingFixMessage* llfix::FixClient< Transport >::outgoing_message_instance ( )  [inline]
```

Obtain a reusable outgoing FIX message instance.

Returns

Pointer to an OutgoingFixMessage.

The returned message is owned by the session

3.2.2.18 process()

```
template<typename Transport >
void llfix::FixClient< Transport >::process ( )  [inline]
```

Process incoming and outgoing FIX protocol activity.

This method drives FIX message handling including:

- Incoming message parsing
- Admin command processing
- Heartbeats and test requests
- Resend requests and gap fills

Call repeatedly when running without an internal thread.

3.2.2.19 send_client_heartbeat()

```
template<typename Transport >
virtual bool llfix::FixClient< Transport >::send_client_heartbeat (
            FixString * test_request_id )  [inline], [protected], [virtual]
```

Send FIX Heartbeat message.

Parameters

test_request_id	Optional TestReqID.

Returns

true if sent successfully.

3.2.2.20 send_gap_fill_message()

```
template<typename Transport >
virtual bool llfix::FixClient< Transport >::send_gap_fill_message ( )  [inline], [protected],
[virtual]
```

Send FIX Gap Fill message.

Returns

true if sent successfully.

3.2.2.21 send_logon_request()

```
template<typename Transport >
virtual bool llfix::FixClient< Transport >::send_logon_request ( )  [inline], [protected],
[virtual]
```

Send FIX Logon request.

Returns

true if sent successfully.

3.2.2.22 send_logout_request()

```
template<typename Transport >
virtual bool llfix::FixClient< Transport >::send_logout_request ( )  [inline], [protected],
[virtual]
```

Send FIX Logout request.

Returns

true if sent successfully.

3.2.2.23 send_outgoing_message()

```
template<typename Transport >
virtual bool llfix::FixClient< Transport >::send_outgoing_message (
            OutgoingFixMessage * message )  [inline], [virtual]
```

Encode and send an outgoing FIX message.

Parameters

message	Outgoing FIX message to send.

Returns

true if the message was successfully sent, false otherwise.

3.2.2.24 send_resend_request()

```
template<typename Transport >
virtual bool llfix::FixClient< Transport >::send_resend_request ( )  [inline], [protected],
[virtual]
```

Send FIX Resend Request message.

Returns

true if sent successfully.

3.2.2.25 send_sequence_reset_message()

```
template<typename Transport >
virtual bool llfix::FixClient< Transport >::send_sequence_reset_message (
            uint32_t desired_sequence_no )  [inline], [protected], [virtual]
```

Send FIX Sequence Reset message.

Parameters

desired_sequence_no	New sequence number.

Returns

true if sent successfully.

3.2.2.26 send_test_request()

```
template<typename Transport >
virtual bool llfix::FixClient< Transport >::send_test_request ( )  [inline], [protected],
[virtual]
```

Send FIX Test Request message.

Returns

true if sent successfully.

3.2.2.27 set_message_persist_plugin()

```
template<typename Transport >
void llfix::FixClient< Transport >::set_message_persist_plugin (
            MessagePersistPlugin * plugin )  [inline]
```

Set the message persistence plugin.

Parameters

plugin	Pointer to a MessagePersistPlugin implementation.

3.2.2.28 shutdown()

```
template<typename Transport >
void llfix::FixClient< Transport >::shutdown (
            bool graceful_shutdown = true )  [inline]
```

Shutdown the FIX client.

Parameters

graceful_shutdown	If true, performs FIX logout handshake.

In threaded mode, this signals the thread to exit. In non-threaded mode, shutdown is performed immediately.

3.2.2.29 specify_repeating_group()

```
template<typename Transport >
template<typename...  Args>
void llfix::FixClient< Transport >::specify_repeating_group (
            Args...  args )  [inline]
```

Specify repeating group definitions for incoming FIX messages.

Template Parameters

Args	Parameter pack defining repeating group structure.

Parameters

args	Repeating group specification arguments.

Not needed in the commercial edition

3.2.2.30 start()

```
template<typename Transport >
bool llfix::FixClient< Transport >::start ( )  [inline]
```

Start the client execution thread.

Returns

true if the thread was successfully created, false otherwise.

When started, the client will execute run() in a loop until shutdown is requested.

The documentation for this class was generated from the following file:

- include/llfix/fix_client.h

3.3 llfix::FixedPoint Class Reference

Represents an unsigned fixed-point numeric value with a configurable number of decimal points. Not general purpose, intended usage is for only exchange connectivity.

`#include <fixed_point.h>`

Public Member Functions

- void set_decimal_points (uint32_t n)

  Set the number of decimal points for this value.
- uint32_t get_decimal_points () const

  Get the number of decimal points for this value.
- void set_raw_value (uint64_t raw_value)

  Set the raw integer value.
- uint64_t get_raw_value () const

  Get the raw integer value.
- void set_from_chars (const char *buffer, std::size_t length)

  Set value from a character buffer representing a decimal number.
- std::size_t to_chars (char *buffer) const

  Convert the fixed-point value into characters in buffer.
- std::string to_string () const

  Convert the fixed-point value to a std::string.
- bool operator== (const FixedPoint &other) const

  Equality comparison.
- bool operator!= (const FixedPoint &other) const

  Inequality comparison.
- bool operator> (const FixedPoint &other) const

  Greater-than comparison.
- bool operator>= (const FixedPoint &other) const

  Greater-than-or-equal comparison.
- bool operator< (const FixedPoint &other) const

  Less-than comparison.
- bool operator<= (const FixedPoint &other) const

  Less-than-or-equal comparison.

3.3.1 Detailed Description

Represents an unsigned fixed-point numeric value with a configurable number of decimal points. Not general purpose, intended usage is for only exchange connectivity.

- Decimal points represent the scale factor. For example: in `10.32`, decimal points = 2.
- If decimal points is 4, a raw value of `10000` represents `1.0000`.
- Some common decimal point examples used by exchanges:
  - 4 (Nasdaq OMX)
  - 6 (Euronext)
  - 7 (Euronext)
  - 8 (Xetra & Eurex)
- Raw value is stored as a 64-bit unsigned integer, matching typical exchange-native binary protocol fields (e.g. price).
- Arithmetic operations not supported. You can use get_raw_value and set_raw_value to modify the underlying value

Warning

1. You must not use a FixedPoint instance without setting decimal points.
2. set_from_chars assumes a strictly valid unsigned decimal representation. Accepted format: digits with an optional single '.' delimiter (e.g. "123", "123.45"). The parser does not validate or skip whitespace, signs, separators, exponent notation, or multiple '.' characters. Any non-digit (other than a single '.') or multiple '.' occurrences yield undefined/implementation-specific results.

3.3.2 Usage

3.3.2.1 Example 1

```
FixedPoint fp;
fp.set_decimal_points(4);
fp.set_from_chars("1000.0023");
std::cout « fp.to_string();      // 1000.0023
std::cout « fp.get_raw_value();  // 10000023
```

3.3.2.2 Example 2

```
FixedPoint fp;
fp.set_decimal_points(4);
fp.set_raw_value(10001234);
std::cout « fp.to_string();     // 1000.1234
std::cout « fp.get_raw_value(); // 10001234
```

3.3.3 Member Function Documentation

3.3.3.1 get_decimal_points()

```
uint32_t llfix::FixedPoint::get_decimal_points ( ) const  [inline]
```

Get the number of decimal points for this value.

Returns

Number of decimal points

3.3.3.2 get_raw_value()

```
uint64_t llfix::FixedPoint::get_raw_value ( ) const  [inline]
```

Get the raw integer value.

Returns

Raw fixed-point value

3.3.3.3 set_decimal_points()

```
void llfix::FixedPoint::set_decimal_points (
            uint32_t n )  [inline]
```

Set the number of decimal points for this value.

Parameters

n	Number of decimal points (max allowed value is 9)

3.3.3.4 set_from_chars()

```
void llfix::FixedPoint::set_from_chars (
            const char * buffer,
            std::size_t length )  [inline]
```

Set value from a character buffer representing a decimal number.

Example: if decimal points = 4, input "10.2345" -> raw value = 102345

Parameters

buffer	Character buffer containing the numeric value
length	Length of the buffer

3.3.3.5 set_raw_value()

```
void llfix::FixedPoint::set_raw_value (
            uint64_t raw_value )  [inline]
```

Set the raw integer value.

Parameters

raw_value	The raw value corresponding to the fixed-point representation

3.3.3.6 to_chars()

```
std::size_t llfix::FixedPoint::to_chars (
            char * buffer ) const  [inline]
```

Convert the fixed-point value into characters in buffer.

Example: raw value 102345 with decimal points 4 -> buffer "10.2345"

Parameters

buffer	Target buffer to write characters

Returns

Number of characters written

Note

The output is NOT null-terminated; callers must append '\0' if a C-string is required.

3.3.3.7 to_string()

```
std::string llfix::FixedPoint::to_string ( ) const  [inline]
```

Convert the fixed-point value to a std::string.

Returns

String representation including decimal point

The documentation for this class was generated from the following file:

- include/llfix/electronic_trading/common/fixed_point.h

3.4 llfix::FixServer< Transport > Class Template Reference

FIX server implementation.

`#include <fix_server.h>`

Inherits Transport, and ManagedInstance.

Public Member Functions

- bool create (const std::string &server_name, const std::string server_config_file_path)
  Creates and initialises the FIX server instance.
- bool add_session (const std::string &session_name, FixSessionSettings &session_settings)
  Adds a FIX session using preloaded session settings.
- bool add_session (const std::string &session_config_file_path, const std::string &session_name)
  Adds a FIX session by loading settings from a configuration file.
- bool add_sessions_from (const std::string &session_config_file_path)
  Adds all FIX sessions found in a configuration file.
- std::size_t get_session_count () const
  Returns the number of configured FIX sessions.
- FixSession * get_session (const std::string &session_name) override
  Retrieves a FIX session by name.
- std::string get_session_name (FixSession *session)
  Retrieves the name of a FIX session.
- void get_session_names (std::vector< std::string > &target) override
  Retrieves the names of all configured FIX sessions.
- template<typename... Args>
  void specify_repeating_group (Args... args)
  Specify repeating group definitions for incoming FIX messages for all sessions.
- OutgoingFixMessage * outgoing_message_instance (FixSession *session)
  Retrieves the reusable outgoing FIX message instance for a session.
- virtual bool send_outgoing_message (FixSession *session, OutgoingFixMessage *message)
  Encodes and sends an outgoing FIX message.
- void shutdown ()
  Shuts down the FIX server.
- virtual void on_new_order (FixSession *session, const IncomingFixMessage *message)
  Called when a New Order (35=D) message is received.
- virtual void on_cancel_order (FixSession *session, const IncomingFixMessage *message)
  Called when an Order Cancel Request (35=F) message is received.
- virtual void on_replace_order (FixSession *session, const IncomingFixMessage *message)
  Called when an Order Cancel/Replace Request (35=G) message is received.

- virtual void on_application_level_reject (FixSession *session, const IncomingFixMessage *message)

  Called when an application-level reject message is generated.
- virtual void on_session_level_reject (FixSession *session, const IncomingFixMessage *message)

  Called when a session-level reject message is generated.
- virtual void on_custom_message (FixSession *session, const IncomingFixMessage *message)

  Called when any other FIX message type is received.
- virtual void on_logon_request (FixSession *session, const IncomingFixMessage *message)

  Called when a Logon (35=A) request is received from a client.
- virtual void on_logout_request (FixSession *session, const IncomingFixMessage *message)

  Called when a Logout (35=5) request is received from a client.
- virtual void on_client_resend_request (FixSession *session, const IncomingFixMessage *message)

  Called when a Resend Request (35=2) is received from a client.
- virtual void on_client_test_request (FixSession *session, const IncomingFixMessage *message)

  Called when a Test Request (35=1) is received from a client.
- virtual void on_client_heartbeat (FixSession *session)

  Called when a Heartbeat (35=0) message is received from a client.
- virtual bool process_incoming_throttling (FixSession *session, const IncomingFixMessage *incoming_fix_↩ message)

  Applies incoming message throttling logic.
- virtual void process_schedule_validator (FixSession *session)

  Validates the session against configured schedule rules.
- virtual FixSession * accept_session (std::size_t peer_index, const IncomingFixMessage *incoming_fix_↩ message, const char *buffer, std::size_t buffer_length, uint32_t parser_reject_code)

  Accepts and validates an incoming session before logon processing.
- virtual void process_logon_request (FixSession *session, std::size_t peer_index, const IncomingFixMessage *message)

  Processes an incoming Logon (35=A) request.
- virtual bool authenticate_logon_request (FixSession *session, const IncomingFixMessage *message)

  Authenticates an incoming Logon (35=A) request.
- void set_message_persist_plugin (MessagePersistPlugin *plugin)

  Set the message persistence plugin.

Protected Member Functions

- virtual bool send_heartbeat (FixSession *session, FixString *test_request_id)

  Sends a Heartbeat (35=0) message to the client.
- virtual bool send_logon_response (FixSession *session, const IncomingFixMessage *message)

  Sends a Logon (35=A) response message.
- virtual bool send_logout_message (FixSession *session, const std::string &reason_text="")

  Sends a Logout (35=5) message.
- virtual bool send_test_request (FixSession *session)

  Sends a Test Request (35=1) message.
- virtual bool send_resend_request (FixSession *session)

  Sends a Resend Request (35=2) message.
- virtual bool send_sequence_reset_message (FixSession *session, uint32_t desired_sequence_no)

  Sends a Sequence Reset (35=4) message without gap fill.
- virtual bool send_gap_fill_message (FixSession *session)

  Sends a Gap Fill message in response to a Resend Request.

3.4.1 Detailed Description

template<typename Transport>
class llfix::FixServer< Transport >

FIX server implementation.

Manages FIX sessions, accepts incoming client connections, routes FIX messages, and provides virtual callbacks for application-level and session-level message handling.

Template Parameters

Transport	Transport layer implementation: one of TcpReactor TcpReactorScalable TcpReactorScalableSSL.

3.4.2 Member Function Documentation

3.4.2.1 accept_session()

```
template<typename Transport >
virtual FixSession* llfix::FixServer< Transport >::accept_session (
            std::size_t peer_index,
            const IncomingFixMessage * incoming_fix_message,
            const char * buffer,
            std::size_t buffer_length,
            uint32_t parser_reject_code )  [inline], [virtual]
```

Accepts and validates an incoming session before logon processing.

Handles parser-level validation failures, checks required header tags, resolves the target session from CompIDs and BeginString, and rejects invalid or duplicate live logon attempts.

Parameters

peer_index	Transport peer index associated with the connection.
incoming_fix_message	Parsed incoming FIX message.
buffer	Raw incoming FIX bytes for logging/reject context.
buffer_length	Length of the raw incoming FIX buffer.
parser_reject_code	Parser validation result code.

Returns

Pointer to the resolved FIX session, or nullptr if rejected.

3.4.2.2 add_session() [1/2]

```
template<typename Transport >
bool llfix::FixServer< Transport >::add_session (
            const std::string & session_config_file_path,
            const std::string & session_name )  [inline]
```

Adds a FIX session by loading settings from a configuration file.

Parameters

session_config_file_path	Path to the session configuration file.
session_name	Name of the session group in the configuration.

Returns

true if the session was added successfully, false otherwise.

3.4.2.3 add_session() [2/2]

```
template<typename Transport >
bool llfix::FixServer< Transport >::add_session (
            const std::string & session_name,
            FixSessionSettings & session_settings )  [inline]
```

Adds a FIX session using preloaded session settings.

Parameters

session_name	Logical name of the session.
session_settings	Session configuration settings.

Returns

true if the session was added successfully, false otherwise.

3.4.2.4 add_sessions_from()

```
template<typename Transport >
bool llfix::FixServer< Transport >::add_sessions_from (
            const std::string & session_config_file_path )  [inline]
```

Adds all FIX sessions found in a configuration file.

Scans the configuration file for groups containing "SESSION" in their name and creates a FIX session for each.

Parameters

session_config_file_path	Path to the session configuration file.

Returns

true if at least one session was added successfully, false otherwise.

3.4.2.5 authenticate_logon_request()

```
template<typename Transport >
virtual bool llfix::FixServer< Transport >::authenticate_logon_request (
            FixSession * session,
            const IncomingFixMessage * message )  [inline], [virtual]
```

Authenticates an incoming Logon (35=A) request.

Allows applications to implement custom authentication logic such as logon usernames and password

Parameters

session	FIX session attempting to log on.
message	Incoming Logon FIX message.

Returns

true if authentication succeeds, false to reject the logon.

3.4.2.6 create()

```
template<typename Transport >
bool llfix::FixServer< Transport >::create (
            const std::string & server_name,
            const std::string server_config_file_path )  [inline]
```

Creates and initialises the FIX server instance.

Parameters

server_name	Logical name of the FIX server instance.
server_config_file_path	Path to the server configuration file.

Returns

true on successful creation, false on error.

3.4.2.7 get_session()

```
template<typename Transport >
FixSession* llfix::FixServer< Transport >::get_session (
            const std::string & session_name )  [inline], [override]
```

Retrieves a FIX session by name.

Parameters

session_name	Name of the FIX session.

Returns

Pointer to the FIX session, or nullptr if not found.

3.4.2.8 get_session_count()

```
template<typename Transport >
std::size_t llfix::FixServer< Transport >::get_session_count ( ) const  [inline]
```

Returns the number of configured FIX sessions.

Returns

Number of active FIX sessions.

3.4.2.9 get_session_name()

```
template<typename Transport >
std::string llfix::FixServer< Transport >::get_session_name (
            FixSession * session )  [inline]
```

Retrieves the name of a FIX session.

Parameters

session	Pointer to a FIX session.

Returns

Session name if found, empty string otherwise.

3.4.2.10 get_session_names()

```
template<typename Transport >
void llfix::FixServer< Transport >::get_session_names (
            std::vector< std::string > & target )  [inline], [override]
```

Retrieves the names of all configured FIX sessions.

Parameters

target	Output vector populated with session names.

3.4.2.11 on_application_level_reject()

```
template<typename Transport >
virtual void llfix::FixServer< Transport >::on_application_level_reject (
            FixSession * session,
            const IncomingFixMessage * message )  [inline], [virtual]
```

Called when an application-level reject message is generated.

Parameters

session	FIX session associated with the reject.
message	Incoming FIX message that caused the reject.

3.4.2.12 on_cancel_order()

```
template<typename Transport >
virtual void llfix::FixServer< Transport >::on_cancel_order (
            FixSession * session,
            const IncomingFixMessage * message )  [inline], [virtual]
```

Called when an Order Cancel Request (35=F) message is received.

Parameters

session	FIX session that received the message.
message	Incoming FIX message.

3.4.2.13 on_client_heartbeat()

```
template<typename Transport >
virtual void llfix::FixServer< Transport >::on_client_heartbeat (
            FixSession * session )  [inline], [virtual]
```

Called when a Heartbeat (35=0) message is received from a client.

Parameters

session	FIX session that received the heartbeat.

3.4.2.14 on_client_resend_request()

```
template<typename Transport >
virtual void llfix::FixServer< Transport >::on_client_resend_request (
            FixSession * session,
            const IncomingFixMessage * message )  [inline], [virtual]
```

Called when a Resend Request (35=2) is received from a client.

Parameters

session	FIX session that received the resend request.
message	Incoming Resend Request FIX message.

3.4.2.15 on_client_test_request()

```
template<typename Transport >
virtual void llfix::FixServer< Transport >::on_client_test_request (
            FixSession * session,
            const IncomingFixMessage * message )  [inline], [virtual]
```

Called when a Test Request (35=1) is received from a client.

Parameters

session	FIX session that received the test request.
message	Incoming Test Request FIX message.

3.4.2.16 on_custom_message()

```
template<typename Transport >
virtual void llfix::FixServer< Transport >::on_custom_message (
            FixSession * session,
            const IncomingFixMessage * message )  [inline], [virtual]
```

Called when any other FIX message type is received.

Parameters

session	FIX session that received the message.
message	Incoming FIX message.

3.4.2.17 on_logon_request()

```
template<typename Transport >
virtual void llfix::FixServer< Transport >::on_logon_request (
            FixSession * session,
            const IncomingFixMessage * message )  [inline], [virtual]
```

Called when a Logon (35=A) request is received from a client.

Parameters

session	FIX session requesting logon.
message	Incoming Logon FIX message.

3.4.2.18 on_logout_request()

```
template<typename Transport >
virtual void llfix::FixServer< Transport >::on_logout_request (
            FixSession * session,
            const IncomingFixMessage * message )  [inline], [virtual]
```

Called when a Logout (35=5) request is received from a client.

Parameters

session	FIX session requesting logout.
message	Incoming Logout FIX message.

3.4.2.19 on_new_order()

```
template<typename Transport >
virtual void llfix::FixServer< Transport >::on_new_order (
            FixSession * session,
            const IncomingFixMessage * message )  [inline], [virtual]
```

Called when a New Order (35=D) message is received.

Parameters

session	FIX session that received the message.
message	Incoming FIX message.

3.4.2.20 on_replace_order()

```
template<typename Transport >
virtual void llfix::FixServer< Transport >::on_replace_order (
            FixSession * session,
            const IncomingFixMessage * message )  [inline], [virtual]
```

Called when an Order Cancel/Replace Request (35=G) message is received.

Parameters

session	FIX session that received the message.
message	Incoming FIX message.

3.4.2.21 on_session_level_reject()

```
template<typename Transport >
virtual void llfix::FixServer< Transport >::on_session_level_reject (
            FixSession * session,
            const IncomingFixMessage * message )  [inline], [virtual]
```

Called when a session-level reject message is generated.

Parameters

session	FIX session associated with the reject.
message	Incoming FIX message that caused the reject.

3.4.2.22 outgoing_message_instance()

```
template<typename Transport >
OutgoingFixMessage* llfix::FixServer< Transport >::outgoing_message_instance (
            FixSession * session )  [inline]
```

Retrieves the reusable outgoing FIX message instance for a session.

Parameters

session	Target FIX session.

Returns

Pointer to the outgoing FIX message instance.

3.4.2.23 process_incoming_throttling()

```
template<typename Transport >
virtual bool llfix::FixServer< Transport >::process_incoming_throttling (
            FixSession * session,
            const IncomingFixMessage * incoming_fix_message )  [inline], [virtual]
```

Applies incoming message throttling logic.

Determines whether an incoming FIX message should be processed, delayed, rejected, or cause session termination based on throttling configuration.

Parameters

session	FIX session receiving the message.
incoming_fix_message	Incoming FIX message.

Returns

true if processing should continue, false otherwise.

3.4.2.24 process_logon_request()

```
template<typename Transport >
virtual void llfix::FixServer< Transport >::process_logon_request (
            FixSession * session,
            std::size_t peer_index,
            const IncomingFixMessage * message )  [inline], [virtual]
```

Processes an incoming Logon (35=A) request.

Validates and authenticates the incoming logon, applies heartbeat timing from the request, sends the logon response, and finalises session state transition to logged on.

Parameters

session	FIX session handling the logon request.
peer_index	Transport peer index associated with the connection.
message	Incoming Logon FIX message.

3.4.2.25 process_schedule_validator()

```
template<typename Transport >
virtual void llfix::FixServer< Transport >::process_schedule_validator (
            FixSession * session )  [inline], [virtual]
```

Validates the session against configured schedule rules.

Checks whether the current time is within the session's allowed schedule and terminates the connection if it is not.

Parameters

session	FIX session to validate.

3.4.2.26 send_gap_fill_message()

```
template<typename Transport >
virtual bool llfix::FixServer< Transport >::send_gap_fill_message (
            FixSession * session )  [inline], [protected], [virtual]
```

Sends a Gap Fill message in response to a Resend Request.

Parameters

session	FIX session to send the gap fill on.

Returns

true if the message was sent successfully, false otherwise.

3.4.2.27 send_heartbeat()

```
template<typename Transport >
virtual bool llfix::FixServer< Transport >::send_heartbeat (
            FixSession * session,
            FixString * test_request_id )  [inline], [protected], [virtual]
```

Sends a Heartbeat (35=0) message to the client.

Parameters

session	FIX session to send the heartbeat on.
test_request_id	Optional TestReqID (tag 112), may be null.

Returns

true if the message was sent successfully, false otherwise.

3.4.2.28 send_logon_response()

```
template<typename Transport >
virtual bool llfix::FixServer< Transport >::send_logon_response (
            FixSession * session,
            const IncomingFixMessage * message )  [inline], [protected], [virtual]
```

Sends a Logon (35=A) response message.

Parameters

session	FIX session to send the logon response on.
message	Incoming Logon FIX message being responded to.

Returns

true if the message was sent successfully, false otherwise.

3.4.2.29 send_logout_message()

```
template<typename Transport >
virtual bool llfix::FixServer< Transport >::send_logout_message (
            FixSession * session,
            const std::string & reason_text = "" )  [inline], [protected], [virtual]
```

Sends a Logout (35=5) message.

Parameters

session	FIX session to send the logout message on.
reason_text	Optional human-readable logout reason.

Returns

true if the message was sent successfully, false otherwise.

3.4.2.30 send_outgoing_message()

```
template<typename Transport >
virtual bool llfix::FixServer< Transport >::send_outgoing_message (
            FixSession * session,
            OutgoingFixMessage * message )  [inline], [virtual]
```

Encodes and sends an outgoing FIX message.

Parameters

session	Target FIX session.
message	Outgoing FIX message to send.

Returns

true if the message was sent successfully, false otherwise.

3.4.2.31 send_resend_request()

```
template<typename Transport >
virtual bool llfix::FixServer< Transport >::send_resend_request (
            FixSession * session )  [inline], [protected], [virtual]
```

Sends a Resend Request (35=2) message.

Parameters

session	FIX session to send the resend request on.

Returns

true if the message was sent successfully, false otherwise.

3.4.2.32 send_sequence_reset_message()

```
template<typename Transport >
virtual bool llfix::FixServer< Transport >::send_sequence_reset_message (
```

```
        FixSession * session,
        uint32_t desired_sequence_no )  [inline], [protected], [virtual]
```

Sends a Sequence Reset (35=4) message without gap fill.

Parameters

session	FIX session to send the sequence reset on.
desired_sequence_no	Target sequence number.

Returns

true if the message was sent successfully, false otherwise.

3.4.2.33 send_test_request()

```
template<typename Transport >
virtual bool llfix::FixServer< Transport >::send_test_request (
            FixSession * session )  [inline], [protected], [virtual]
```

Sends a Test Request (35=1) message.

Parameters

session	FIX session to send the test request on.

Returns

true if the message was sent successfully, false otherwise.

3.4.2.34 set_message_persist_plugin()

```
template<typename Transport >
void llfix::FixServer< Transport >::set_message_persist_plugin (
            MessagePersistPlugin * plugin )  [inline]
```

Set the message persistence plugin.

Parameters

plugin	Pointer to a MessagePersistPlugin implementation.

3.4.2.35 shutdown()

```
template<typename Transport >
void llfix::FixServer< Transport >::shutdown ( )  [inline]
```

Shuts down the FIX server.

Stops the underlying transport and terminates all sessions.

3.4.2.36 specify_repeating_group()

```
template<typename Transport >
template<typename... Args>
void llfix::FixServer< Transport >::specify_repeating_group (
            Args... args ) [inline]
```

Specify repeating group definitions for incoming FIX messages for all sessions.

Template Parameters

Args	Parameter pack defining repeating group structure.

Parameters

args	Repeating group specification arguments.

Not needed in the commercial edition

The documentation for this class was generated from the following file:

- include/llfix/fix_server.h

3.5 llfix::FixSession Class Reference

Represents a single FIX protocol session.

```
#include <fix_session.h>
```

Inherits ManagedInstanceSession.

Public Member Functions

- std::string get_name () const override

  Returns the logical name of the FIX session.
- SessionState get_state ()

  Retrieves the current state of the FIX session.
- SequenceStore * get_sequence_store ()

  Provides access to the session's sequence store.
- template<typename T >
  bool add_attribute (const std::string &attribute, const T &value)

  Adds or updates a session-scoped attribute.
- bool get_attribute (const std::string &attribute, std::string &value) const

  Retrieves a session attribute value.

3.5.1 Detailed Description

Represents a single FIX protocol session.

FixSession encapsulates the full lifecycle, state management, sequencing, throttling, validation, and message handling logic for a FIX connection.

3.5.2 Member Function Documentation

3.5.2.1 add_attribute()

```
template<typename T >
bool llfix::FixSession::add_attribute (
            const std::string & attribute,
            const T & value )  [inline]
```

Adds or updates a session-scoped attribute.

Attributes are stored as key/value string pairs and can be used to associate arbitrary metadata with a session instance.

Supported value types:

- std::string
- Arithmetic types (converted via std::to_string)
- Any type supporting stream insertion (operator<<)

Template Parameters

T	Attribute value type.

Parameters

attribute	Attribute name (key).
value	Attribute value.

Returns

true if the attribute was added successfully, false otherwise.

3.5.2.2 get_attribute()

```
bool llfix::FixSession::get_attribute (
            const std::string & attribute,
            std::string & value ) const  [inline]
```

Retrieves a session attribute value.

Looks up an attribute by name and returns its value if present.

Parameters

attribute	Attribute name (key).
value	Output parameter receiving the attribute value.

Returns

true if the attribute exists, false otherwise.

3.5.2.3 get_name()

```
std::string llfix::FixSession::get_name ( ) const  [inline], [override]
```

Returns the logical name of the FIX session.

The session name uniquely identifies this FixSession instance and is typically configured during initialisation.

Returns

Session name as a std::string.

3.5.2.4 get_sequence_store()

```
SequenceStore* llfix::FixSession::get_sequence_store ( )  [inline]
```

Provides access to the session's sequence store.

The sequence store maintains incoming and outgoing FIX message sequence numbers and persists them according to configuration.

Returns

Pointer to the underlying SequenceStore.

3.5.2.5 get_state()

```
SessionState llfix::FixSession::get_state ( )  [inline]
```

Retrieves the current state of the FIX session.

The returned value represents the lifecycle and connection state of the session. Possible states are:

- SessionState::NONE Session has not been initialised.
- SessionState::DISABLED Session is administratively disabled.
- SessionState::DISCONNECTED Session is not connected to the peer.
- SessionState::LOGGED_OUT Session is disconnected after a successful logout.
- SessionState::LOGON_REJECTED Logon attempt was rejected by the peer.
- SessionState::PENDING_CONNECTION TCP connection established, waiting to initiate logon.
- SessionState::PENDING_LOGON Logon message sent, awaiting logon response.
- SessionState::PENDING_LOGOUT Logout message sent or received, awaiting completion.
- SessionState::LOGGED_ON Session is logged on and fully operational.
- SessionState::IN_RETRANSMISSION_INITIATED_BY_SELF Session is retransmitting messages initiated by this side.
- SessionState::IN_RETRANSMISSION_INITIATED_BY_PEER Session is retransmitting messages initiated by the peer.

Returns

Current SessionState.

The documentation for this class was generated from the following file:

- include/llfix/fix_session.h

3.6 llfix::FixUtilities Class Reference

Utility functions for working with FIX messages.

```
#include <fix_utilities.h>
```

Static Public Member Functions

- static std::string fix_to_human_readible (const char *buffer, std::size_t buffer_length)

  Converts a FIX message buffer to a human-readable string.

3.6.1 Detailed Description

Utility functions for working with FIX messages.

Contains static helper functions

3.6.2 Member Function Documentation

3.6.2.1 fix_to_human_readible()

```
static std::string llfix::FixUtilities::fix_to_human_readible (
            const char * buffer,
            std::size_t buffer_length )  [inline], [static]
```

Converts a FIX message buffer to a human-readable string.

Replaces the FIX delimiter character with '|'.

Parameters

buffer	Pointer to the FIX message buffer.
buffer_length	Length of the buffer.

Returns

Human-readable string representation of the FIX message.

The documentation for this class was generated from the following file:

- include/llfix/fix_utilities.h

3.7 llfix::IncomingFixMessage Class Reference

Represents a parsed incoming FIX message.

```
#include <incoming_fix_message.h>
```

Public Member Functions

- bool has_tag (uint32_t tag) const

  Checks whether a FIX tag exists and is valid.
- bool has_repeating_group_tag (uint32_t tag) const

  Checks whether a repeating group tag exists.
- std::string to_string () const

Serialises the FIX message into a human-readable string.

- template<typename T >
  T get_tag_value_as (uint32_t tag, std::size_t decimal_points=0) const

  Retrieves a FIX tag value converted to the requested type.
- template<typename T >
  T get_repeating_group_tag_value_as (uint32_t tag, std::size_t index, std::size_t decimal_points=0) const

  Retrieves a repeating group tag value converted to the requested type.

3.7.1 Detailed Description

Represents a parsed incoming FIX message.

Always call has_tag() or has_repeating_group_tag() before accessing values.

Repeating groups are handled separately via IncomingFixRepeatingGroups and accessed through dedicated APIs.

3.7.2 Member Function Documentation

3.7.2.1 get_repeating_group_tag_value_as()

```
template<typename T >
T llfix::IncomingFixMessage::get_repeating_group_tag_value_as (
            uint32_t tag,
            std::size_t index,
            std::size_t decimal_points = 0 ) const  [inline]
```

Retrieves a repeating group tag value converted to the requested type.

Template Parameters

T	Target return type.

Parameters

tag	FIX tag number.
index	Repeating group index (0-based).
decimal_points	Number of decimal places (required for floating-point and FixedPoint).

Returns

Tag value converted to type T.

Note

Supported types:

- std::string_view
- std::string
- char
- bool
- integral types
- floating-point types
- FixedPoint

3.7.2.2 get_tag_value_as()

```
template<typename T >
T llfix::IncomingFixMessage::get_tag_value_as (
            uint32_t tag,
            std::size_t decimal_points = 0 ) const  [inline]
```

Retrieves a FIX tag value converted to the requested type.

Template Parameters

T	Target return type.

Parameters

tag	FIX tag number.
decimal_points	Number of decimal points (required for floating-point and FixedPoint types).

Returns

Tag value converted to type T.

Note

Supported types:

- std::string_view
- std::string
- char
- bool
- integral types
- floating-point types
- FixedPoint

3.7.2.3 has_repeating_group_tag()

```
bool llfix::IncomingFixMessage::has_repeating_group_tag (
            uint32_t tag ) const  [inline]
```

Checks whether a repeating group tag exists.

Parameters

tag	FIX tag number belonging to a repeating group.

Returns

true if the tag exists in the repeating group storage, false otherwise.

3.7.2.4 has_tag()

```
bool llfix::IncomingFixMessage::has_tag (
            uint32_t tag ) const  [inline]
```

Checks whether a FIX tag exists and is valid.

Parameters

tag	FIX tag number.

Returns

true if the tag exists, false otherwise.

3.7.2.5 to_string()

```
std::string llfix::IncomingFixMessage::to_string ( ) const  [inline]
```

Serialises the FIX message into a human-readable string.

The output is formatted as:

- Header fields first (8, 9, 35, 34, 49, 52, 56)
- Body fields (excluding header and trailer)
- Repeating group fields
- Trailer field (10)

Fields are separated by the '|' character instead of SOH.

Returns

A string representation of the FIX message

The documentation for this class was generated from the following file:

- include/llfix/incoming_fix_message.h

3.8 llfix::ManagementServer Class Reference

Management TCP server for FIX engine runtime control and inspection.

`#include <management_server.h>`

Inherits TcpReactor< Epoll >.

Public Member Functions

- bool register_client (ManagedInstance *instance)

  Registers a managed FIX client instance with the management server.
- bool register_server (ManagedInstance *instance)

  Registers a managed FIX server instance with the management server.

3.8.1 Detailed Description

Management TCP server for FIX engine runtime control and inspection.

ManagementServer provides a lightweight, delimiter-based TCP interface used to manage and introspect FIX clients and FIX servers at runtime.

It is **not** a Telnet-compatible server and accepts only plain TCP connections. Commands and responses are delimited using `Commands::COMMAND_DELIMITER` (e.g. '|').

The server supports:

- Runtime registration of managed FIX client and server instances
- Execution of management commands via CommandFactory
- Centralised access to engine metadata (version, start time, log path)

Note

The server must be successfully created via create() before registering any managed instances.

Warning

Instance names must be unique across both clients and servers.

3.8.2 Member Function Documentation

3.8.2.1 register_client()

```
bool llfix::ManagementServer::register_client (
            ManagedInstance * instance )  [inline]
```

Registers a managed FIX client instance with the management server.

Adds the provided ManagedInstance to the internal client registry, making it accessible to management commands

The management server must be successfully initialised prior to calling this method.

Parameters

instance	Pointer to a ManagedInstance representing a FIX client.

Returns

true if the client was successfully registered, false otherwise.

Note

Client instance names must be unique across all registered clients and servers.

3.8.2.2 register_server()

```
bool llfix::ManagementServer::register_server (
            ManagedInstance * instance )  [inline]
```

Registers a managed FIX server instance with the management server.

Adds the provided ManagedInstance to the internal server registry, allowing management commands to interact with FIX server components.

The management server must be fully initialised before invoking this method.

Parameters

instance	Pointer to a ManagedInstance representing a FIX server.

Returns

true if the server was successfully registered, false otherwise.

Note

Server instance names must be unique across all registered clients and servers.

The documentation for this class was generated from the following file:

- include/llfix/management_server/management_server.h

3.9 llfix::MessagePersistPlugin Class Reference

Interface for custom FIX message persistence plugins.

```
#include <message_persist_plugin.h>
```

Public Member Functions

- virtual void persist_incoming_message (const std::string_view &session_name, uint32_t sequence_number, const char ∗buffer, std::size_t buffer_size)=0

  Persists an incoming FIX message.
- virtual void persist_outgoing_message (const std::string_view &session_name, uint32_t sequence_number, const char ∗buffer, std::size_t buffer_size, bool successfully_transmitted)=0

  Persists an outgoing FIX message.

3.9.1 Detailed Description

Interface for custom FIX message persistence plugins.

MessagePersistPlugin allows users to implement custom persistence logic for FIX messages

To create a custom message persister, derive from this interface and implement both persist methods:

- persist_incoming_message()
- persist_outgoing_message()

The plugin can be enabled by calling:

- llfix::FixClient::set_message_persist_plugin(), or
- llfix::FixServer::set_message_persist_plugin()

3.9.2 Member Function Documentation

3.9.2.1 persist_incoming_message()

```
virtual void llfix::MessagePersistPlugin::persist_incoming_message (
            const std::string_view & session_name,
            uint32_t sequence_number,
            const char * buffer,
            std::size_t buffer_size )  [pure virtual]
```

Persists an incoming FIX message.

This method is invoked for every FIX message received by the engine

Parameters

session_name	Logical FIX session name.
sequence_number	FIX message sequence number.
buffer	Pointer to the raw FIX message buffer.
buffer_size	Size of the FIX message buffer in bytes.

Note

The buffer is only valid for the duration of this call and must not be stored directly without copying.

3.9.2.2 persist_outgoing_message()

```
virtual void llfix::MessagePersistPlugin::persist_outgoing_message (
            const std::string_view & session_name,
            uint32_t sequence_number,
            const char * buffer,
            std::size_t buffer_size,
            bool successfully_transmitted )  [pure virtual]
```

Persists an outgoing FIX message.

This method is invoked for every FIX message sent by the engine. The persistence callback includes information about whether the message was successfully transmitted.

Parameters

session_name	Logical FIX session name.
sequence_number	FIX message sequence number.
buffer	Pointer to the raw FIX message buffer.
buffer_size	Size of the FIX message buffer in bytes.
successfully_transmitted	Indicates whether the message was successfully sent to the peer.

Note

The buffer is only valid for the duration of this call and must not be stored directly without copying.

The documentation for this class was generated from the following file:

- include/llfix/electronic_trading/common/message_persist_plugin.h

3.10 llfix::OutgoingFixMessage Class Reference

FIX message builder and encoder for outbound messages.

```
#include <outgoing_fix_message.h>
```

Public Member Functions

- void set_msg_type (char c)

  Sets FIX MsgType (tag 35) using a single character.
- void set_msg_type (std::string_view buffer)

  Sets FIX MsgType (tag 35) using a string view.

- template<FixMessageComponent component = FixMessageComponent::BODY, typename T >
  void set_tag (uint32_t tag, T val, std::size_t decimal_points=0)

  Appends a FIX tag to the specified message component.
- template<FixMessageComponent component = FixMessageComponent::BODY>
  void set_binary_tag (uint32_t tag, const char *buffer, std::size_t data_length)

  Appends a FIX tag using the provided buffer with raw data.
- template<FixMessageComponent component = FixMessageComponent::BODY>
  void set_timestamp_tag (uint32_t tag)

  Appends a FIX timestamp tag using the current session time.
- std::string get_sending_time ()

  Returns the currently cached tag 52 (SendingTime) value.

3.10.1 Detailed Description

FIX message builder and encoder for outbound messages.

3.10.2 Member Function Documentation

3.10.2.1 get_sending_time()

```
std::string llfix::OutgoingFixMessage::get_sending_time ( )  [inline]
```

Returns the currently cached tag 52 (SendingTime) value.

Returns

SendingTime as a string.

3.10.2.2 set_binary_tag()

```
template<FixMessageComponent component = FixMessageComponent::BODY>
void llfix::OutgoingFixMessage::set_binary_tag (
            uint32_t tag,
            const char * buffer,
            std::size_t data_length )  [inline]
```

Appends a FIX tag using the provided buffer with raw data.

Template Parameters

component	FIX message component to append to (default: FixMessageComponent::BODY)

Parameters

tag	FIX tag number representing a binary field
buffer	start address of buffer that hold raw/binary data
data_length	length of buffer

3.10.2.3 set_msg_type() [1/2]

```
void llfix::OutgoingFixMessage::set_msg_type (
            char c )  [inline]
```

Sets FIX MsgType (tag 35) using a single character.

This overload is intended for FIX message types represented by a single character (e.g. 'D', '8', '0').

Parameters

c	FIX MsgType character

3.10.2.4 set_msg_type() [2/2]

```
void llfix::OutgoingFixMessage::set_msg_type (
            std::string_view buffer )  [inline]
```

Sets FIX MsgType (tag 35) using a string view.

Supports multi-character FIX message types (e.g. "AE", "XLR").

Parameters

buffer	String view containing the message type

Precondition

Maximum buffer length is 4 characters

3.10.2.5 set_tag()

```
template<FixMessageComponent component = FixMessageComponent::BODY, typename T >
void llfix::OutgoingFixMessage::set_tag (
            uint32_t tag,
```

```
            T val,
            std::size_t decimal_points = 0 )  [inline]
```

Appends a FIX tag to the specified message component.

Encodes the supplied value into an internal FixString and appends it to the selected FIX message component (HEADER, BODY, or TRAILER).

3.10.2.5.1 Supported Value Types

- const char*
- std::string
- std::string_view
- char
- bool (Y/N)
- FixedPoint
- Integral types (signed / unsigned)
- Floating-point types (requires decimal_points)

Template Parameters

component	FIX message component to append to (default: FixMessageComponent::BODY)
T	Value type to encode

Parameters

tag	FIX tag number
val	Value to encode
decimal_points	Number of decimal places for floating-point values

3.10.2.6 set_timestamp_tag()

```
template<FixMessageComponent component = FixMessageComponent::BODY>
void llfix::OutgoingFixMessage::set_timestamp_tag (
            uint32_t tag )  [inline]
```

Appends a FIX timestamp tag using the current session time.

Encodes the current timestamp once per message using the session's configured sub-second precision and reuses it for all timestamp tags within the same message.

Template Parameters

component	FIX message component to append to (default: FixMessageComponent::BODY)

Parameters

tag	FIX tag number representing a timestamp field

The documentation for this class was generated from the following file:

- include/llfix/outgoing_fix_message.h

3.11 llfix::SequenceStore Class Reference

Persistent FIX sequence number store backed by a memory-mapped file.

`#include <sequence_store.h>`

Public Member Functions

- void reset_numbers ()

  Reset both incoming and outgoing sequence numbers to zero.
- void set_outgoing_seq_no (uint32_t n)

  Set the outgoing FIX sequence number.
- uint32_t get_outgoing_seq_no () const

  Get the outgoing FIX sequence number.
- void set_incoming_seq_no (uint32_t n)

  Set the incoming FIX sequence number.
- uint32_t get_incoming_seq_no () const

  Get the incoming FIX sequence number.

3.11.1 Detailed Description

Persistent FIX sequence number store backed by a memory-mapped file.

Maintains incoming and outgoing FIX sequence numbers and persists them to disk using a memory-mapped file to survive process restarts.

3.11.2 Member Function Documentation

3.11.2.1 get_incoming_seq_no()

`uint32_t llfix::SequenceStore::get_incoming_seq_no ( ) const [inline]`

Get the incoming FIX sequence number.

Returns

Current incoming sequence number

3.11.2.2 get_outgoing_seq_no()

```
uint32_t llfix::SequenceStore::get_outgoing_seq_no ( ) const  [inline]
```

Get the outgoing FIX sequence number.

Returns

Current outgoing sequence number

3.11.2.3 set_incoming_seq_no()

```
void llfix::SequenceStore::set_incoming_seq_no (
            uint32_t n )  [inline]
```

Set the incoming FIX sequence number.

Parameters

n	New incoming sequence number

3.11.2.4 set_outgoing_seq_no()

```
void llfix::SequenceStore::set_outgoing_seq_no (
            uint32_t n )  [inline]
```

Set the outgoing FIX sequence number.

Parameters

n	New outgoing sequence number

The documentation for this class was generated from the following file:

- include/llfix/electronic_trading/session/sequence_store.h

Index