# The MDF .NET API

Technical Description and Reference Guide

8 December 2015

# Installation

The MDF .NET API can be downloaded via anonymous ftp from: **ftp://ftp.millistream.com**. It's a Class Library built as a single assembly, MDF.dll (which you add a reference to in your .NET project). It's built for "Any CPU" (meaning it will JIT compile to either x64 or x86 at runtime). The MDF .NET API wraps the MDF C/C++ API which in turn links against zlib (http://zlib.net) and OpenSSL (http://openssl.org/). So in addition to the MDF.dll you also have to download the Windows binaries of libmdf (the MDF C/C++ API). You need three DLL files from it: zlib1.dll, libeay32.dll, libmdf-0.dll (located in the bin and bin64 folders).

Full source code is also available for download for those wanting to compile and build on their own. The source code package contains solution and project files for Visual Studio 2015. In order to be able to build you will also need a header file from the libmdf package, called mdf_fields.h (located in the include\mdf folder). Put this in the same folder as the HeaderDefinesConverter.tt file.

# The Millistream_MDF namespace

The entire API exists in the `Millistream_MDF` namespace. It consists of three classes: `MDF`, `MDFUtils` and `mdf_fields`, and two objects `MDFData` and `MDFGetDataResult`. You can make the easily accessible by adding a `using` directive at the top of your file:

```
using Millistream_MDF;
```

# The MDF class

This is the main class that has methods for connecting, requesting and receiving data. An instance/object of this class is created using the empty constructor:

```
MDF mdf = new MDF();
```

## Connecting to the Millistream System

The mdf object can attempt to connect and logon to the Millistream System using the `connect()` method. Supply a comma separated list of servers and the API will try each server in turn until it finds one that answers.

After successful connection, it will try to logon using the supplied username and password. If both the connection and logon is successful, the `connect()` method returns true. In all other cases it returns false.

The example below will attempt to connect to the server "sandbox.millistream.com" at tcp port "9100".

```
bool result = mdf.connect("sandbox.millistream.com:9100", "username", "password");
```

The connection can be manually disconnected using the `disconnect()` method, any current connection will also automatically be disconnected when calling the `destroy()` method.

It is also possible to register a status delegate, which will be called by the API during the various stages of the connect procedure (and when/if the connection is disconnected):

```
void StatusDelegate(MDF.MDF_CONN_STATUS status, string host, string ip) { ... }
...

mdf.StatusDelegateMethod = StatusDelegate;
```

## Requesting Data

The Millistream Data Feed is a request based feed, so upon a successful login, there will be no data sent by the server unless the client requests them. Requests can be of two types; stream or image.

Stream requests (`mdf_fields.MDF_RT_STREAM`) are realtime streaming data, I.e when an event happens at a market place, the event is immediately sent to the client by the server in realtime. The first data message received for an instrument after a stream request is with only the fields that where changed due to the event.

Image requests (`mdf_fields.MDF_RT_IMAGE`) are requests for the current image of the instruments. There will only be a single reply for each instrument in an image request and it will be a complete data message with all the fields that currently contain a value (null values will be sent if they have been deliberately set to null by a stream event).

Usually, clients will issue combined requests (i.e. stream+image) since they want to know the current image of the instruments and also subscribe to the realtime changes to this image. This is known as a full request (`mdf_fields.MDF_RT_FULL`).

Currently the request functionality of the Millistream system is somewhat limited (for example there is no possibility to request trades or news messages sent previous to the request, and there is not possibility to request only instruments of a specific type or belonging to a specific market or list), but this is something that is subject to change soon due to active development in this particular field.

The filtering possibility in the current version of the system is to request all available instruments or to supply a space separated list of the instruments to subscribe to. For example, a client application could request Basic Data for all instruments and then issue requests based upon the received Basic Data, this works quite well since Basic Data is a low messages per second stream.

Issuing requests is done using the `request()` method and below is an example on how to issue requests for both Basic Data and Quotes, the request is a full request and for all available instruments:

```
mdf.request(mdf_fields.MDF_RC_BASICDATA + " " + mdf_fields.MDF_RC_QUOTE, mdf_fields.MDF_RT_FULL);
```

When issuing full requests, each instrument will be opened in turn, I.e first the image will be sent and then the stream subscription will be enabled on that instrument. So the first message for every instrument in a full request will always be a complete message, and each “opened” instrument will begin to send it's realtime stream as soon as possible meaning that the client will not have to wait for the whole image part to complete before receiving realtime data.

Trades and news does currently not contain image data, so full requests for these messages will be handled like stream requests only.

If the 5th parameter of the `request()` method, `onlyUpdatesSinceDisconnect` is true, it will request only data updated since the last message was received from the MDF feed. This can be useful for example on reconnect where a client does not want to perform a complete download of all the messages and fields in order to limit the data needed to be received and processed.

When the client no longer wants to subscribe to realtime data it can issue a `fields.MDF_M_UNSUBSCRIBE` request to unsubscribe to the specified data and instruments.

## Receiving Data

Data is received from the MDF feed in the form of messages. The .NET API converts these messages into .NET data objects for your convenience.

Receiving data can be done in two ways:

1. Using a delegate method.

```
void DataDelegate(List<MDFData> dataList) {...}
...
mdf.DataDelegateMethod = DataDelegate;
```

`mdf.startReceivingDataUsingDelegate();`

2. Using `mdf.getData()`

   `MDFGetDataResult result = mdf.getData();`

   The `MDFGetDataResult` object consists of

   - `RESULT_CODE resultCode` is an enum that can have the following values:
     - `NEW_DATA` reflects that `getData()` returned data
     - `NO_DATA` reflects that `getData()` did **not** return any data (dataList is null)
     - `DISCONNECTED` reflects that connection to the Millistream server has been lost.
   - `List<MDFData> dataList` (a `List` of `MDFData)`, explained below.

   `mdf.getData()` should be called repeatedly in order to receive real time data of stream requests.

# MDFData

The `MDFData` is the .NET representation of an MDF message. It consists of:

- `uint insref` – is the instrument reference, and is the unique id of the instrument within the Millistream universe. An instrument should never change instrument reference, and instrument references will never be reused. There are a limited number of messages in which `insref` will not be used to carry the instrument reference, please consult the *MDF Messages Reference* document for more information.
- `int mref` - the message reference. Tells which type of data is in the `fields` `Dictionary`. This is the value to match with the `MDF_M_` constants in the `mdf_fields` class.
- `int mclass` - the message class is normally only used internally by Millistream and is supplied to the client for completeness and transparency. It will match the `MDF_MC_` constants in the `mdf_fields` class. The client should under most circumstances only use the message reference in order to determine which message it has received.
- `Dictionary<uint, string> fields` - a key/value dictionary of data fields.

  The field keys can be matched with the field definition constants `MDF_F_` in the `mdf_fields` class to determine which data field it represents.

  The field value is a UTF-8 string which can be null if the field value is supposed to be null which is a valid value in the Millistream Data Feed.

# mdf_fields (constants)

The `mdf_fields` class is a .NET adaptation of **`mdf_fields.h`** from the MDF C/C++ API and only contains (a large number of) constants. These constants represent definitions of messages, field tags etc.

### Definition prefixes

| Prefix | Description |
|---|---|
| MDF_M_ | Message References |
| MDF_MC_ | Message Classes |

| | |
|---|---|
| MDF_RC_ | Request Classes |
| MDF_RT_ | Request Types |
| MDF_F_ | Fields / Tags |
| MDF_CA_ | Corporate Action Types |
| MDF_TC_ | Trade Codes |
| MDF_OPT_ | API Options |
| MDF_ERR_ | API Errors |
| MDF_STATUS_ | API Status Callback Status Codes |

So for example, an `MDFData` object's `mref` element might have the value equal to `mdf_fields.MDF_M_QUOTE` to indicate that its `fields` `Dictionary` contains quote data. In that case, `fields` might contain a key that equals `mdf_fields.MDF_F_BIDPRICE` which indicates it is a Bid Price field. A detailed description of the available messages is documented in the *MDF Messages Reference* document, and the fields are documented in the *MDF Fields Reference* document.

## Wrapped C API functions

All functions in the MDF C/C++ API are wrapped by the MDF .NET API and made public which gives you access to them. This might be beneficial if you are used to working with the C/C++ API or if you need advanced/custom functionality that the .NET API does not provide. It requires that you understand how to use IntPtr, the Marshal class and procedural programming.

None of these functions are included in the methods reference below, instead we refer to the separate MDF C/C++ API documentation.

# MDF()

**SYNOPSIS**

```
public MDF()
```

**DESCRIPTION**

The constructor of the MDF class. Initializes a new instance of the MDF class.

**RETURN VALUE**

An instance of the MDF class is returned.

**ERRORS**

None

# connect()

## SYNOPSIS

```
public bool connect(string servers, string username, string password)
```

## DESCRIPTION

Attempts to connect and logon to the specified MDF server(s).

`servers` - a comma separated list of 'host:port' pairs, where 'host' can be a DNS host name or an IP address (IPv6 addressed must be enclosed in brackets). If the first server in the list does not respond in time, the next server in the list will be tried until the list is empty and the function finally fails.

Upon connect, the API will verify the authenticity of the server using its public RSA key, and a secure channel will be set up between the client and the server before the function signals success.

`username` – your account user name provided by Millistream

`password` – your account password provided by Millistream

## RETURN VALUE

Returns true if connect and logon was successful. Otherwise it returns false.

## ERRORS

| | |
|---|---|
| **`MDF_ERR_ARGUMENT`** | The `servers` argument was missing or contained erroneous data |
| **`MDF_ERR_CONNECTED`** | The API handle is already connected, if you want to reconnect, you must first disconnect the current connection. |
| **`MDF_ERR_CONNECT`** | The connection attempt failed with every server on the list |

# request()

**SYNOPSIS**

```
public void request(string requestClass, string requestType, string insRefs = null,
                    bool onlyUpdatesSinceDisconnect = false)
```

**DESCRIPTION**

Requests data from the Millistream Data Feed.

`requestClass` - a space separated list that specifies the type of data to request. A description of all available request classes is documented in the *MDF Fields Reference* document, by the definition of MDF_F_REQUESTCLASS.

`requestType -` specifies the type of request and can be either stream, image or full (i.e. both image and stream).

`insRefs` - a space separated list of insrefs that specifies which instruments to request data for. If omitted or `null`, it will request data for all available instruments.

`onlyUpdatesSinceDisconnect` – specifies whether or not to request only data updated since the last message was received from the MDF feed. This can be used when reconnecting after a disconnect to avoid receiving data that you already received prior to the disconnect.

**RETURN VALUE**

None

**ERRORS**

None

# getData()

**SYNOPSIS**

```
public MDFGetDataResult getData()
```

**DESCRIPTION**

Attempts to receive data from the MDF feed. If there currently is no data to receive, the function waits for **DataTimeout** number of seconds, if **DataTimeout** is zero (0) the function will return immediately.

**RETURN VALUE**

The `MDFGetDataResult` object consists of

- `RESULT_CODE resultCode` is an enum that can have the following values:
    - `NEW_DATA` reflects that `getData()` returned data
    - `NO_DATA` reflects that `getData()` did **not** return any data (dataList is null)
    - `DISCONNECTED` reflects that connection to the Millistream server has been lost.
- `List<MDFData> dataList` is a `List` of `MDFData`

The `MDFData` is the .NET representation of an MDF message. It consists of:

- `uint insref` – is the instrument reference, and is the unique id of the instrument within the Millistream universe. An instrument should never change instrument reference, and instrument references will never be reused. There are a limited number of messages in which `insref` will not be used to carry the instrument reference, please consult the *MDF Messages Reference* document for more information.
- `int mref` - the message reference. Tells which type of data is in the `fields` `Dictionary`. This is the value to match with the `MDF_M_` constants in the `mdf_fields` class.
- `int mclass` - the message class is normally only used internally by Millistream and is supplied to the client for completeness and transparency. It will match the `MDF_MC_` constants in the `mdf_fields` class. The client should under most circumstances only use the message reference in order to determine which message it has received.
- `Dictionary<uint, string> fields` - a key/value dictionary of data fields.

  The field keys can be matched with the field definition constants `MDF_F_` in the `mdf_fields` class to determine which data field it represents.

  The field value is a UTF-8 string which can be null if the field value is supposed to be null which is a valid value in the Millistream Data Feed.

**ERRORS**

| | |
|---|---|
| **`MDF_ERR_NOT_CONNECTED`** | There is not connection with a server. |
| **`MDF_ERR_DISCONNECTED`** | We were disconnected. |

| | |
|---|---|
| **`MDF_ERR_CONNECTION_IDLE`** | There has been no timely reply to any of our heartbeats so the connection has been disconnected. |
| **`MDF_ERR_MSG_OOB`** | The received message contained lengths that would overflow the message |
| **`MDF_ERR_NO_MEM`** | There was not sufficient available memory to fulfill a call to **`realloc()`** when resizing the consume buffer to accommodate for the incoming message. |
| **`MDF_ERR_NO_ERROR`** | No errors occurred when decoding the message |
| **`MDF_ERR_AUTHFAIL`** | The authentication of the message failed due to either communications errors or an injection attempt. The connection with the server has been dropped. |
| **`MDF_ERR_UNKNOWN_TEMPLATE`** | The current message contains a message reference for which the API does not know the template, so the message cannot be decoded. |
| **`MDF_ERR_TEMPLATE_OOB`** | The current field is outside the template defined for this message, either the API templates are out of date, or we have received some errenous data. |

## startReceivingDataUsingDelegate()

**SYNOPSIS**

```
public void startReceivingDataUsingDelegate()
```

**DESCRIPTION**

Starts receiving data from the MDF feed and calls the `DataDelegateMethod` as soon as new data has been received. **This should only be used when you have set the `DataDelegateMethod` property.**

**RETURN VALUE**

None

**ERRORS**

None

# stopReceivingDataUsingDelegate()

## SYNOPSIS

```
public void startReceivingDataUsingDelegate()
```

## DESCRIPTION

Stops receiving data from the MDF. This should only be used if you have called `startReceivingDataUsingDelegate().`

## RETURN VALUE

None

## ERRORS

None

# disconnect()

## SYNOPSIS

```
public void disconnect()
```

## DESCRIPTION

Logs off and disconnects from the server. Safe to call even if the mdf object is already disconnected.

## RETURN VALUE

None

## ERRORS

None

## destroy()

**SYNOPSIS**

```
public void destroy()
```

**DESCRIPTION**

Destroys the handle of the wrapped C/C++ API. Any open connection will be automatically closed. All memory allocated by the handle will be freed.

**RETURN VALUE**

None

**ERRORS**

None

# DataTimeout

**SYNOPSIS**

```
public int DataTimeout { get; set; }
```

**DESCRIPTION**

Gets or sets the maximum number of seconds the getData() method should wait to receive data before it returns (if there currently is no data to receive). If set to zero (0), getData() will always return immediately. Default value is 10.

**RETURN VALUE**

None

**ERRORS**

None

# Error

## SYNOPSIS

```
public MDF_ERROR Error { get; }
```

## DESCRIPTION

Gets the current API error code. Can be `MDF_ERR_NO_ERROR`, `MDF_ERR_NO_MEM`, `MDF_ERR_MSG_OOB`, `MDF_ERR_TEMPLATE_OOB`, `MDF_ERR_UNKNOWN_TEMPLATE`, `MDF_ERR_ARGUMENT`, `MDF_ERR_CONNECTED`, `MDF_ERR_NOT_CONNECTED`, `MDF_ERR_CONNECT`, `MDF_ERR_MSG_TO_LARGE`, `MDF_ERR_CONNECTION_IDLE`, `MDF_ERR_DISCONNECTED` or `MDF_ERR_AUTHFAIL`

## RETURN VALUE

The current API error code

## ERRORS

None

# ReceivedBytes

## SYNOPSIS

```
public UInt64 ReceivedBytes { get; }
```

## DESCRIPTION

Gets the total number of bytes received from the MDF feed.

## RETURN VALUE

None

## ERRORS

None

# SentBytes

**SYNOPSIS**

```
public UInt64 SentBytes { get; }
```

**DESCRIPTION**

Gets the total number of bytes sent to the MDF feed.

**RETURN VALUE**

None

**ERRORS**

None

# StatusDelegateMethod

## SYNOPSIS

```
public StatusDelegate StatusDelegateMethod { get; set; }
```

## DESCRIPTION

Gets or sets the status delegate method.

## RETURN VALUE

None

## ERRORS

None

## DataDelegateMethod

**SYNOPSIS**

```
public DataDelegate DataDelegateMethod { get; set; }
```

**DESCRIPTION**

Gets or sets the data delegate method.

**RETURN VALUE**

None

**ERRORS**

None

# MDFHandle

## SYNOPSIS

```
public IntPtr MDFHandle { get; }
```

## DESCRIPTION

Gets the handle to the C/C++ API. Should only be used when implementing functionality which the MDF class doesn’t have.

## RETURN VALUE

None

## ERRORS

None

# MessagePointer

## SYNOPSIS

```
public IntPtr MessagePointer { get; }
```

## DESCRIPTION

Gets the pointer to the message struct that is used to send messages via the C/C++ API to the Millistream server. Should only be used when implementing functionality which the MDF class doesn’t have.

## RETURN VALUE

None

## ERRORS

None

# StatusDelegate()

## SYNOPSIS

```
public delegate void StatusDelegate(MDF_CONN_STATUS status, string host, string ip)
```

## DESCRIPTION

Represents the method that is called whenever there is a change of the status of the connection.

`status` – the new status of the connection. Can be either `MDF_STATUS_LOOKUP`, `MDF_STATUS_CONNECTING`, `MDF_STATUS_CONNECTED` or `MDF_STATUS_DISCONNECTED`

`host` – the DNS host name for which the connection status has changed

`ip` – the ip address for which the connection status has changed

## RETURN VALUE

None

## ERRORS

None

# DataDelegate()

## SYNOPSIS

```
public delegate void DataDelegate(List<MDFData> dataList)}
```

## DESCRIPTION

Represents the method that is called whenever new data is received from the MDF feed (but only if you have set a `DataDelegateMethod` and called `startReceivingDataUsingDelegate()`).

`dataList` is the list of data received from the feed. See the method `getData()` above for a definition of `MDFData`.

## RETURN VALUE

None

## ERRORS

None