MTConnect C++ Agent Version 1.1.0.8

The C++ Agent provides the a complete implementation of the HTTP server required by the MTConnect standard. The agent provides the protocol and data collection framework that will work as a standalone server. Once built, you only need to specify the XML description of the devices and the location of the adapter.

Version 1.1.0.8 add the ability to run the C++ Agent as a Windows service and support for a configuration file instead of command line arguments. The agent can accept input from a socket in a pipe (|) delimited stream according to the descriptions given in the adapter guide.

The current win32 binary is built statically and requires no additional dlls to run. This allows for a single exe distributable.

Usage

```
agent [help|install|debug|run] [configuration_file]
       help            Prints this message
       install         Installs the service
       remove          Remove the service
       debug           Runs the agent on the command line with
                          verbose logging
       run             Runs the agent on the command line
       config_file     The configuration file to load
                       Default: agent.cfg in current directory
```

When the agent is started without any arguments it is assumed it will be running as a service and will begin the service initialization sequence. The full path to the configuration file is stored in the registry in the following location:

```
\\HKEY_LOCAL_MACHINE\SOFTWARE\MTConnect\MTConnect Agent\ConfigurationFile
```

Directories

`agent/` This contains the main application entry point and the Makefile

`lib/` Third party library source. Contains source for cppunit, dlib, and libxml++.

`samples/` Sample XML configuration files.

`src/` The main source for the app.

`test/` Various unit tests.

`win32/` Libraries required only for the win32 build and the win32 solution.

Windows Binary Release

The windows binary releases come with a prebuilt exe that is statically linked with the Microsoft Runtime libraries. Aside from the standard system libraries, the agent only requires winsock libraries. The agent has been test with version of Windows 2000 and later.

`bin/` Win32 binary (no dependencies required).

Building

Download cmake from http://www.cmake.org/cmake/resources/software.html

Configure cmake using the CMakeLists.txt file in the agent directory. This will generate a project file for the target platform. See CMake documentation for more information.

Configuration

The configuration file is using the standard Boost C++ file format. The configuration file format is flexible and allows for both many adapters to be served from one agent and an adapter to feed multiple agents. The format is:

```
1.   Key = Value
2.
```

The key can only occur once within a section and the value can be any sequence of characters followed by a <CR>. There no significance to the order of the keys, so the file can be specified in free form. We will go over some configurations from a minimal configuration to more complex multi-adapter configurations.

Example 1:

Here's an example configuration file. The # character can be used to comment sections of the file. Everything on the line after the # is ignored. We will start with a extremely simple configuration file.

```
3.   # A very simple file...
4.   Devices = VMC-3Axis.xml
```

This is a one line configuration that specifies the XML file to load for the devices. Since all the values are defaulted, an empty configuration configuration file can be specified. This configuration file will load the VMC-3Axis.xml file and try to connect to an adapter located on the localhost at port 7878. VMC-3Axis.xml must only contain the definition for one device, if more devices exist, and error will be raised and the process will exit.

Example 2:

Most configuration files will specify at least one adapter. The adapters are contained within a block. The Boost configuration file format allows for nested configurations and block associations. There are a number of configurations that can be given for each adapter. Multiple adapters can be specified for one device as well.

```
1.   Devices = VMC-3Axis.xml
2.
3.   Adapters
4.   {
5.       VMC-3Axis
6.       {
7.           Host = 192.168.10.22
8.           Port = 7878 # Default value...
9.       }
10.  }
11.
```

This example loads the devices file as before, but specifies the list of adapters. The device is taken from the name VMC-3Axis that starts the nested block and connects to the adapter on 192.168.10.22 and port 7878. Another way of specifying the equivalent configuration is:

```
12.
1.   Devices = VMC-3Axis.xml
2.
3.   Adapters
4.   {
5.       Adapter_1
6.       {
7.           Device = VMC-3Axis
8.           Host = 192.168.10.22
9.           Port = 7878 # Default value...
10.      }
11.  }
```

Line 7 specifies the Device name associated with this adapter explicitly. We will show how this is used in the next example.

Example 3:

Multiple adapters can supply data to the same device. This is done by creating multiple adapter entries and specifying the same Device for each.

```
1.   Devices = VMC-3Axis.xml
2.
3.   Adapters
4.   {
5.       Adapter_1
6.       {
7.           Device = VMC-3Axis
8.           Host = 192.168.10.22
9.           Port = 7878 # Default value...
10.      }
11.
12.      # Energy sensor
13.      Adapter_2
14.      {
15.          Device = VMC-3Axis
16.          Host = 192.168.10.2
17.          Port = 7878 # Default value...
18.      }
19.  }
```

Both Adapter_1 and Adapter_2 will feed the VMC-3Axis device with different data items. The Adapter_1 name is arbitrary and could just as well be named EnergySensor if desired as illustrated below.

```
1.   Devices = VMC-3Axis.xml
2.
3.   Adapters
4.   {
5.       Controller
6.       {
7.           Device = VMC-3Axis
8.           Host = 192.168.10.22
9.           Port = 7878 # Default value...
10.      }
11.
12.      EnergySensor
13.      {
14.          Device = VMC-3Axis
15.          Host = 192.168.10.2
16.          Port = 7878 # Default value...
17.      }
```

```
18.  }
```

Example 4:

In this example we change the port to 80 which is the default http port.

```
1.   Devices = MyDevices.xml
2.   Port = 80
3.
4.   Adapters
5.   {
6.     ...
```

For browsers you will no longer need to specify the port to connect to.

Example 5:

If multiple devices are specified in the XML file, there must be an adapter feeding each device.

```
1.   Devices = MyDevices.xml
2.
3.   Adapters
4.   {
5.       VMC-3Axis
6.       {
7.           Host = 192.168.10.22
8.       }
9.
10.      HMC-5Axis
11.      {
12.          Host = 192.168.10.24
13.      }
14.  }
```

This will map associate the adapters for these two machines to the VMC and HMC devices in MyDevices.xml file. The ports are defaulted to 7878, so we are not required to specify them.

Example 6:

Logging configuration is specified using the logger_config block. You can change the logging_level to specify the verbosity of the logging as well as the destination of the logging output.

```
1.   logger_config
2.   {
3.       logging_level = debug
4.       output = file debug.log
5.   }
```

This will log everything from debug to fatal to the file debug.log. For only fatal errors you can specify the following:

```
1.   logger_config
2.   {
3.       logging_level = fatal
4.   }
```

The default file is agent.log in the same directory as the agent.exe file resides. The default logging level is info. To have the agent log to the command window:

```
1.   logger_config
2.   {
3.       logging_level = debug
4.       output = cout
5.   }
```

This will log debug level messages to the current console window. When the agent is run with debug, it is sets the logging configuration to debug and outputs to the standard output as specified above.

Configuration Parameters

Top level configuration items

`BufferSize`	The 2^X number of slots available in the circular buffer for samples, events, and conditions. **Default**: 17 ⇒ 2^{17} = 131,072 slots.
`CheckpointFrequency`	The frequency checkpoints are created in the stream. This is used for `current` with the `at` argument. This is an advanced configuration item and should not be changed unless you understand the internal workings of the agent. **Default**: 1000
`Devices`	The XML file to load that specifies the devices and is supplied as the result of a probe request. If the key is not found the defaults are tried. **Defaults**: probe.xml or Devices.xml

`PidFile`	**UNIX only**. The full path of the file that contains the process id of the daemon. This is not supported in Windows. **Default**: agent.pid
`Port`	The port number the agent binds to for requests. **Default**: 5000

Adapter configuration items

`Adapters`	Adapters begins a list of device blocks. If the `Adapters` are not specified and the Devices file only contains one device, a default device entry will be created with an adapter located on the localhost and port 7878 associated with the device in the devices file. **Default:** localhost 5000 associated with the default device
`Device`	The name of the device that corresponds to the name of the device in the Devices file. Each adapter can map to one device. Specifying a "*" will map to the default device. **Default:** The name of the block for this adapter or if that is not found the default device if only one device is specified in the devices file.
`Host`	The host the adapter is located on. **Default:** localhost
`Port`	The port to connect to the adapter. **Default:** 7878
`Manufacturer`	Replaces the manufacturer attribute in the device XML. **Default:** Current value in device XML.
`Station`	Replaces the Station attribute in the device XML. **Default:** Current value in device XML.
`SerialNumber`	Replaces the SerialNumber attribute in the device XML. **Default:** Current value in device XML.
`UUID`	Replaces the UUID attribute in the device XML. **Default:** Current value in device XML.

logger_config configuration items

`logger_config`	The logging configuration section.
`logging_level`	The logging level: trace, debug, info, warn, error, or fatal. **Default:** info

`output`	The output file or stream. If a file is specified specify as: “file <filename>”. cout and cerr can be used to specify the standard output and standard error streams. Defaults to the same directory as the executable. **Default:** file adapter.log