1. Introduction

1.1. Purpose

The Athelon Trading System (ATS) is in prototype form. It's been placed on GitHub for those who need a reference for real-time data access using ActiveTick as a data provider.

The intention of this system was to provide level II real-time data collection and analytic processing for securities trading. It was not meant to be placed directly on datacenter ECNs.

Only two of seven modules were coded and tested in a debug capacity. Subsystems include the Data collection and analytics services.

Development has currently stopped on this project. I hope that the code might be useful for anyone needing a real-time data acquisition example using the ActiveTick provider.

1.2. Requirements

ATS currently runs on Windows and was developed using C++. The solution was designed to run a few of the subsystems on Windows and the rest on RHEL 7/8. The system uses MS SQL server for data storage, and queueing is all implemented in internal data structures. ActiveTick and reading from snapshot files are used as data provider sources.

The intention was to include the capability to choose from multiple sources including snapshot files.

Hardware / Software Requirements (applies to a VM based configuration):

- Windows Server 2019 or Windows 10 (latest updates)
    - 75 GB of disk space
    - 4 GB of memory
    - 2-4 CPU
- MS SQL Server 2016
- MS Visual Studio 2017
- ActiveTick Windows (C++ Based) Distribution (see installation section)
- Get a copy of the ***BareTail*** tool. You'll need a tool such as this to run a tail on the log file.

1.3. Deployment

If you're only interested in getting working example of access to provider level II real-time data (via ActiveTick) then a simple clone of the project will be sufficient.

To deploy a working version, follow the installation instructions below.

1.4. ActiveTick Provider Software

A few ActiveTick components have been included in the source distribution in order for the build to work. They are the property of ActiveTick LLC, and can be downloaded from their Internet Web site.

Please setup a market data developer account and download the distribution. You'll need to do this for the run-time components. (More information on this in the installation section.)

2. Installation

The following are steps you'll need to follow in order to build the project and run it.

If your only interest is to review the code:

- Clone the project on GitHub

If you intend to run the prototype code:

- Meet requirements
- Get ActiveTick libraries
- Configure SQL Server
- Configure Project Folder
- Build the project
- Run the project

2.1. ATS GitHub Source Code

The source code for ATS can be found on GitHub at https://github.com/lbolt/ATS.

2.2. System Requirements

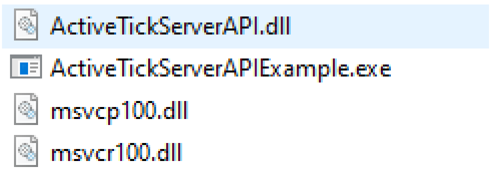
Check the ***Introduction*** section above for details.

2.3. ActiveTick Provider Software

Once you've setup a "market data" account with the provider, log on and download the Market Data API.

I believe the file is still identified as "atfeed-cppsdk-windows.x64.exe". The file can be found at location: https://www.activetick.com/activetick/contents/MarketDataServicesAPIDownload.aspx. This project is based on a 64-bit environment build. I've included a few files in order for the build to succeed – see below.

- Install the package
- Make certain "{your-installation-folder} \ActiveTick Feed API C++ SDK\Bin" is available in the path from which the application will be executed. These are the required files:



ActiveTickServerAPI.dll
ActiveTickServerAPIExample.exe
msvcp100.dll
msvcr100.dll

  - See "misc/bin" for these files in case you did have not yet signed-up for a market data account

I haven't used AT it since mid-2020, I assume it's still available and functional. Don't count on support, they've never responded to any of my emails – ever!

In the end, their data service product is very inexpensive and works really well. If you're a C++ programmer like myself, it's one of the few C++ APIs for real-time data feeds that remains on the market as of this writing.

2.4. MS SQL Server Deployment

I've included an export of an ATS database. You can apply (import) it to your instance to leverage a sample set of watch pool data. This data can be used with the included snapshot file for the file provider.

These are the steps necessary to get this to work.

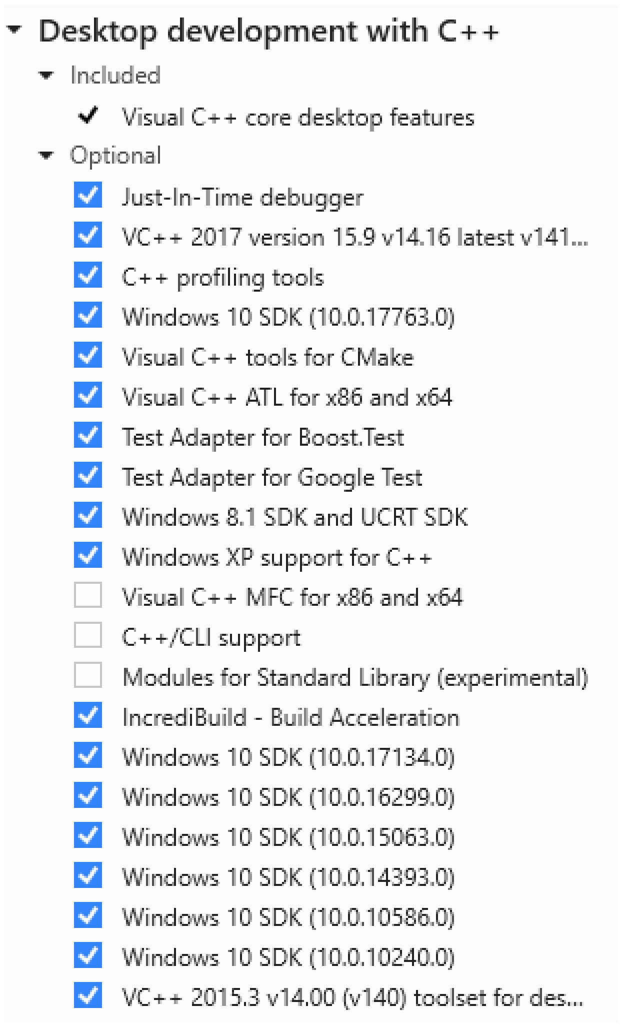
- Log on to SQL Server as "sa"
- Under security, add the user ATS_APP_USER (take defaults), set a preferred password
- Import the data file from the "misc/db" directory from the cloned project
    - Start SQL Server Manager Studio
    - Log on as "sa"
    - Right click on Databases
    - Select Restore Database
        - For source, select device and add the file "misc/db/ATS.bak"
        - For destination, select database and specify "ATS"
        - Select "ok" and the "ATS" database will be restored
- Set ATS_APP_USER as owner of the restored "ATS" database
- Make sure you can log on as ATS_APP_USER with access the ATS database
    - The "dbo.watchpool" table will contain securities that match what's in the snapshot file for testing
- There is no need for a DSN. But I would suggest you setup a system DSN to test the connection. Use "SQL Server" as the source type. This is what is used programmatically.

At this point the database is restored and configured for ATS access.

2.5. Project Build in Visual Studio

Open Visual studio 2017, go to "Tools->Get Tools and Features" and add the following features:

1. Check "Desktop development with C++"
    - Add the following "optional" features:



- Desktop development with C++
  - Included
    - ✓ Visual C++ core desktop features
  - Optional
    - [✓] Just-In-Time debugger
    - [✓] VC++ 2017 version 15.9 v14.16 latest v141...
    - [✓] C++ profiling tools
    - [✓] Windows 10 SDK (10.0.17763.0)
    - [✓] Visual C++ tools for CMake
    - [✓] Visual C++ ATL for x86 and x64
    - [✓] Test Adapter for Boost.Test
    - [✓] Test Adapter for Google Test
    - [✓] Windows 8.1 SDK and UCRT SDK
    - [✓] Windows XP support for C++
    - [ ] Visual C++ MFC for x86 and x64
    - [ ] C++/CLI support
    - [ ] Modules for Standard Library (experimental)
    - [✓] IncrediBuild - Build Acceleration
    - [✓] Windows 10 SDK (10.0.17134.0)
    - [✓] Windows 10 SDK (10.0.16299.0)
    - [✓] Windows 10 SDK (10.0.15063.0)
    - [✓] Windows 10 SDK (10.0.14393.0)
    - [✓] Windows 10 SDK (10.0.10586.0)
    - [✓] Windows 10 SDK (10.0.10240.0)
    - [✓] VC++ 2015.3 v14.00 (v140) toolset for des...

You can probably get away with less, but the Windows 10 SDK features must be there.

Note: It would be advisable to check for Visual Studio updates before moving on.

2. Open “{your-top-level-folder} \ATS\Solution\Code\Ide\Athelon Trading System.sln” solution file with Visual Studio 2017. This may take a minute to complete project loading.
   - Select “Build->Configuration Manager”
   - Select “x64” from the “Active Solution Platform” dropdown and the “Active Solution Configuration” should be set to “DEBUG”
3. In Solution Explorer, right click on the “DataService” project heading and set it as the Startup Project
4. Select “Build->Clean Solution” – Creates the project folders
5. Select “Build->Build Solution” – Debug build
6. **Note**: you must perform configuration steps (next section) prior to running the project

2.6. Configure the Project

Move the provided project files into newly created folders before running the project.

1. **Common configuration steps**:

Add folders under the location housing the project executables. The build steps outlined above produce a debug version of the executables. These files are located in the "{your-top-level-folder}\ATS\Solution\Code\Ide\x64\Debug" folder. Copy the following folders from the cloned project to this location.

Copy folder and contents:

From:

"{your-top-level-folder} \ATS\Solution\Code\Misc\Debug\config}"

To:

"{your-top-level-folder} \ATS\Solution\Code\Ide\x64\Debug\config"

Copy folder and contents:

From:

"{your-top-level-folder} \ATS\Solution\Code\Misc\Debug\Log"

To:

"{your-top-level-folder} \ATS\Solution\Code\Ide\x64\Debug\Log"

Copy folder and contents:

From:

"{your-top-level-folder} \ATS\Solution\Code\Misc\Debug\snapshot"

To:

"{your-top-level-folder} \ATS\Solution\Code\Ide\x64\Debug\snapshot"

2. **Setup the database connection**:
   - Edit file "{your-top-level-folder} \ATS\Solution\Code\Ide\x64\Debug\config\DataService.config"
   - Set the following:

```
<DBManagement
        User="ATS_APP_USER"
        Credential="your-password"
        Server="your-db-server-name"
        Database="ATS"
        Driver="{SQL Server}"
/>
```

   Set your password and database server name.

3. **Configure for snapshot file**:
   - Edit file "{your-top-level-folder} \ATS\Solution\Code\Ide\x64\Debug\config\DataService.config"
   - Set the following:

```
< DataFile_Provider
```

```
Enabled="true"
/>
<AT_Provider
        Enabled="false"
        …
/>
```
Set the enabled keyword.

4. **Configure for ActiveTick Provider**:
   - Edit file "{your-top-level-folder}\ATS\Solution\Code\Ide\x64\Debug\config\DataService.config"
   - Set the following:

```
<AT_Provider
        Enabled="true"
        DisableRDQInsertion="false"
        UserID="your_user_id"
        Credential="your_password"
        GUID_KEY="you_guid_key"
        PrimaryServer="activetick1.activetick.com"
        SecondaryServer="activetick1.activetick.com"
        ServerPort="443"
        UseInternalQueue="false"
        CallbackCycleDelay="0"
        TrackStatsForSymbol="TQQQ"
/>
< DataFile_Provider
        Enabled="false"
/>
```

Set the highlighted fields. You will receive a GUID-KEY upon subscription to the ActiveTick Market Data service. The TrackStatsForYymbol keyword allows the selected symbol to be logged if Trade is enabled under logging.

2.7. Run the Project

Once the project is built and the configuration is set, you may execute the data service project component (you will have previously selected "DataService" as the startup project).

Press ***F5*** in Visual Studio and the project will start. Use the ***BareTail*** utility to run a tail on the logfile located in the “~debug\Log” folder.

Enjoy…

3. Notes

3.1. LTLS Analytic (Linear Trend Long Short)

This analytic algorithm keys on linear tread analysis. The analytic first establishes a linear trend based on regression analysis applied to real-time data collected for a specific security. Once a trend is established the analytic provides notification then goes into a follow state. The follow state is the tradable portion of the established pattern. Once the trend is broken the analytic provides notification and reverts bask to the establish phase to look for a new trend.

This analytic keys on the last trade amount. Since the time skew for the last trade amount can be anywhere from 25 milliseconds (acceptable outside of the data center) to several thousand milliseconds it would make more sense to establish the calculation point somewhere between the bid and ask.

3.2. Coding Practices

The project is coded using uncomplicated C++. Character encoding is UTF-8 / UCS-2. Database characters are wide-character.

I avoided the use of Streams, Lambdas and Double ampersand (&&), Copyable and Movable reference types, etc., to keep things reasonably readable.

3.3. Threading

You can adjust the number of threads for data collection and analytics processing, but I've found under heavy load anything above 4 threads for data collection, and 8 threads for analytics provides no additional benefit.