**Project Title: Weather Logger**

**Introduction**

This article describes the process of downloading and inserting xml file into a database and automating the task using two different approaches i.e. SQL Server Jobs and Windows Task Scheduler.

**Business Case**

[National Oceanic and Atmospheric Administration (NOAA)](http://www.noaa.gov/) provides a wide range of public web services which can be accessed programmatically. Specifically, this [link](http://w1.weather.gov/xml/current_obs/) [1] got my attention. NOAA provides access to observed weather conditions across 1800 locations mostly airports (organized by a station ID) in the United States. Various weather sensors record the values such as barometric pressure, wind speed, humidity, temperature, and so on at these locations. The observations are bundled into a XML file primarily targeted for machine to machine transfer [2]. For each hour, RSS and XML files are generated for every observation station. The suggested pick up time is 15 minutes after the hour. This scenario presented an opportunity to automate the task of downloading the new xml file which is getting updated every hour. The next section describes the overview of the application.

**Overview**

The process is divided into four steps as shown below in figure 1.

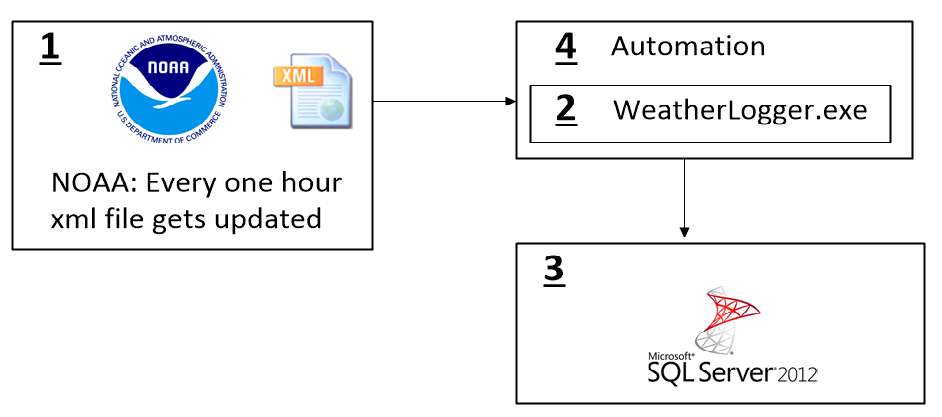


Figure 1: Overview

Before I jump into the implementation details, would like to discuss the boundaries on which the application is designed. The XML file generated by NOAA is about ~2.25 KB. Normally there should be 24 new files every day because the file is getting updated every hour but what I observed is file doesn’t change for hours together. Check out this [link](http://w1.weather.gov/xml/current_obs/KEWR.xml) to see the XML file used for the application and I was interested in the following nodes: ObservationTime, Temp\_F, Temp\_C, Relative Humidity, Wind MPH, pressure\_mb, dewpoint\_F.

The main logic is embedded into the WeatherLogger.exe code which is step two. This is a console based C# application developed in Visual Studio 2012 with .NET 4.5 framework. In the next section I will go over the design strategies incorporated.

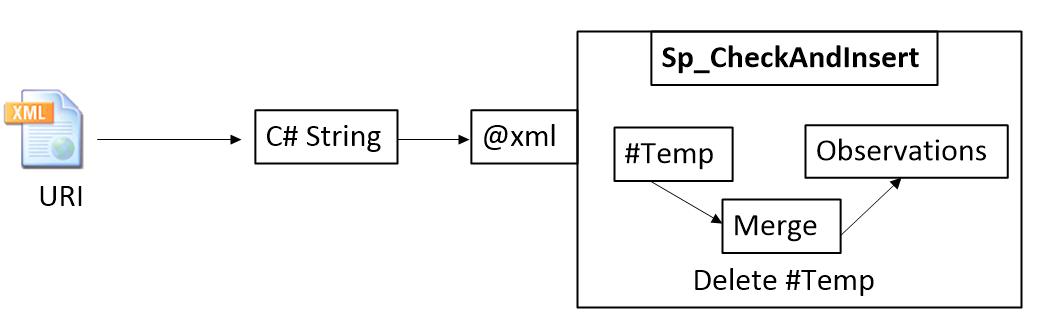
From the beginning I wanted to store the values from the XML file to a table in database. For step three, I wrote two stored procs and created two tables called “Observations” and “AppErrors” in SQL Server 2012.

For the final part, step four, the automation is taken care in two ways. I used windows task scheduler and SQL Server Job to run the WeatherLogger.exe every one hour to pick the new file. Both approaches are good and better when compared with windows service application.

The source code is on GitHub, check out this link. Feel free to pull the code, try it out and I appreciate criticism on the code. If there is room for improvement, would like to know. The next section describes the implementation details of step two.

**WeatherLogger**

This was my initial approach which I scrapped. The heavy lifting was done by the console app and the database portion was light weight. I thought of poling the NOAA server periodically, download the xml file, parse it, remove the unused nodes and insert the nodes required into a table. I was able to achieve all of it but the tricky part was polling the server at a regular interval. I battled through the Last-Modified and If-Last-Modified http headers [2]. Tuned a timing belt which runs at the suggested pick time. Although the suggested pick up time was 15 minutes after the hour, the pickup time was not reliable. This messed up my timing belt, then I resorted for a second option.

For the second option I decided to go database intensive and keep the console app light weight. I heard lot of good things about the xml data type along with the merge statement in SQL Server and decided to use it. This choice significantly reduced the code size and solution became elegant. The main logic is bundled into a stored procedure sp\_CheckAndInsert which takes @xml file as a parameter.

The idea is convert the xml/uri into C# string and call the strod proc by passing the string as an argument. The values from the required nodes are inserted into a #staging table which is identical to the main table which holds the observations data.

select

current\_observation.value('(observation\_time)[1]', 'varchar(50)') as ObsTime,

current\_observation.value('(temp\_f)[1]', 'float') as Temp\_f,

current\_observation.value('(temp\_c)[1]', 'float') as Temp\_c,

current\_observation.value('(relative\_humidity)[1]', ca'float') as RelHum,

current\_observation.value('(wind\_mph)[1]', 'float') as Wind\_mph ,

current\_observation.value('(pressure\_mb)[1]', 'float') as Pressure\_mb,

current\_observation.value('(dewpoint\_f)[1]', 'float') as Dewpt\_f

into #Staging

from

@Raw\_Xml.nodes('current\_observation') as rawdata(current\_observation)

Then use merge tsql command by choosing the Observations as target table and #Staging as source table. Comparison is done using the observation time, which is most reliable column. If not matched then the data from the xml file is inserted into the database. Finally the #Staging table is dropped.

;merge into Observations

using #Staging

on #Staging.ObsTime = Observations.ObsTime

when not matched then

insert (ObsTime,Temp\_f,Temp\_c,RelHum,Wind\_mph,Pressure\_mb,Dewpt\_f)

values (ObsTime,Temp\_f,Temp\_c,RelHum,Wind\_mph,Pressure\_mb,Dewpt\_f);

This approach is scalable and suitable to this scenario as the size of the XML is really low, the execution time for merge statement is very fast. There is no need for storing or parsing the xml file. This stored procedure is called in the InsertData method from the InsertIntoDb class. The class diagram for the console app is shown in figure 2.

There are three main classes in the application, DownloadXML, InserIntoDb and LogErrors, the Program class orchestrates the flow of the application and it is the entry point. The xml data type in SQL Server is quite powerful, this datatype reads xml file as blob and in C# it should be well formatted xml file stored as a simple C# string.

The DownloadWeatherXML method in the DownloadXML uses the WebClient class to download the requested url as a string.

string xml = new WebClient().DownloadString(\_url);

string modifiedxml = xml.Replace("<?xml version=\"1.0\" encoding=\"ISO-8859-1\"?> \r\n", "");

If I directly pass this string without replacing the encoding then I got the following error from the sp\_CheckAndInsert stored procedure. I was unable to resolve it, but it works if I replace the encoding.

“XML parsing: line 1, character 43, unable to switch the encoding”

The LogErrors is generic helper class and has a method InsertErrors(). This method is used for recording any errors that are caught by any catch block throughout the code. The data is inserted by InsertIntoAppErros stored procedure. I was compelled to use this class after reading the book Joel on Software [3].

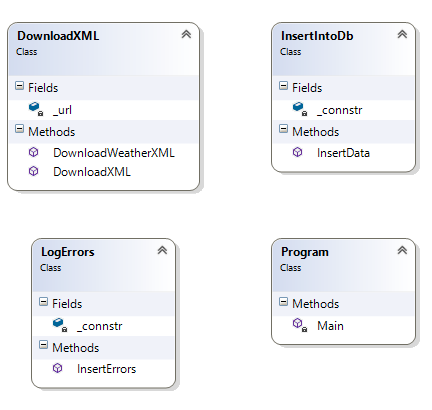


Figure 2: WeatherLogger

**Automation**

The WeatherLogger.exe has to run every hour, which calls in for an automation. I wanted to work with both the SQL Server Job and Windows Task Scheduler. I uploaded the screenshots on how to setup the job, check the links below.

1. Scheduling SQL Server Job
2. Windows Task Scheduler

**Conclusion**

Overall, I personally feel this approach is scalable, the choice of executing the exe can be delegated to the system admin or a database administrator. I have done xml parsing in C# but ease provided by the merge statement and xml data type from SQL Server are unbeatable. For future, I would like to set up a SSRS web service and render the report for a client using the observations data. Also, would like to implement the complete process as SQL CLR stored procedure.

**References**

[1] <http://w1.weather.gov/xml/current_obs/>

[2] <http://products.weather.gov/PDD/NWS_Current_Observations_RSS_XML.pdf>

[3] <http://w1.weather.gov/xml/current_obs/KEWR.xml>

[4] <http://msdn.microsoft.com/en-us/library/system.net.httpwebrequest.ifmodifiedsince.aspx>

[4] <http://www.amazon.com/Joel-Software-Occasionally-Developers-Designers/dp/1590593898>