**Monitis API Wrapper Quick Guide**

**About the Project**

The company Monitis allows registered users and businesses to monitor the performance statistics of a number of IT-related applications and devices including websites, servers, databases and networks.

Registered users are able to perform the monitoring actions or requests using a dashboard control section on the main Monitis website.

<http://monitis.com/>

The aim of this project is to provide a way for users or companies to access the functionality offered by the Monitis dashboard programmatically through the use of various wrapper classes and methods. The project can be extended and customised by users to meet their own preferences and requirements and could also be integrated into a website or IT system.

**Monitis API Documentation**

The Monitis API documentation has been used as a guide for this project and is available at the URL address below. The majority of the information there appears to be accurate although I have personally found that some value types that were recommended to be used for certain variables in monitoring classes did not work as I had expected them to.

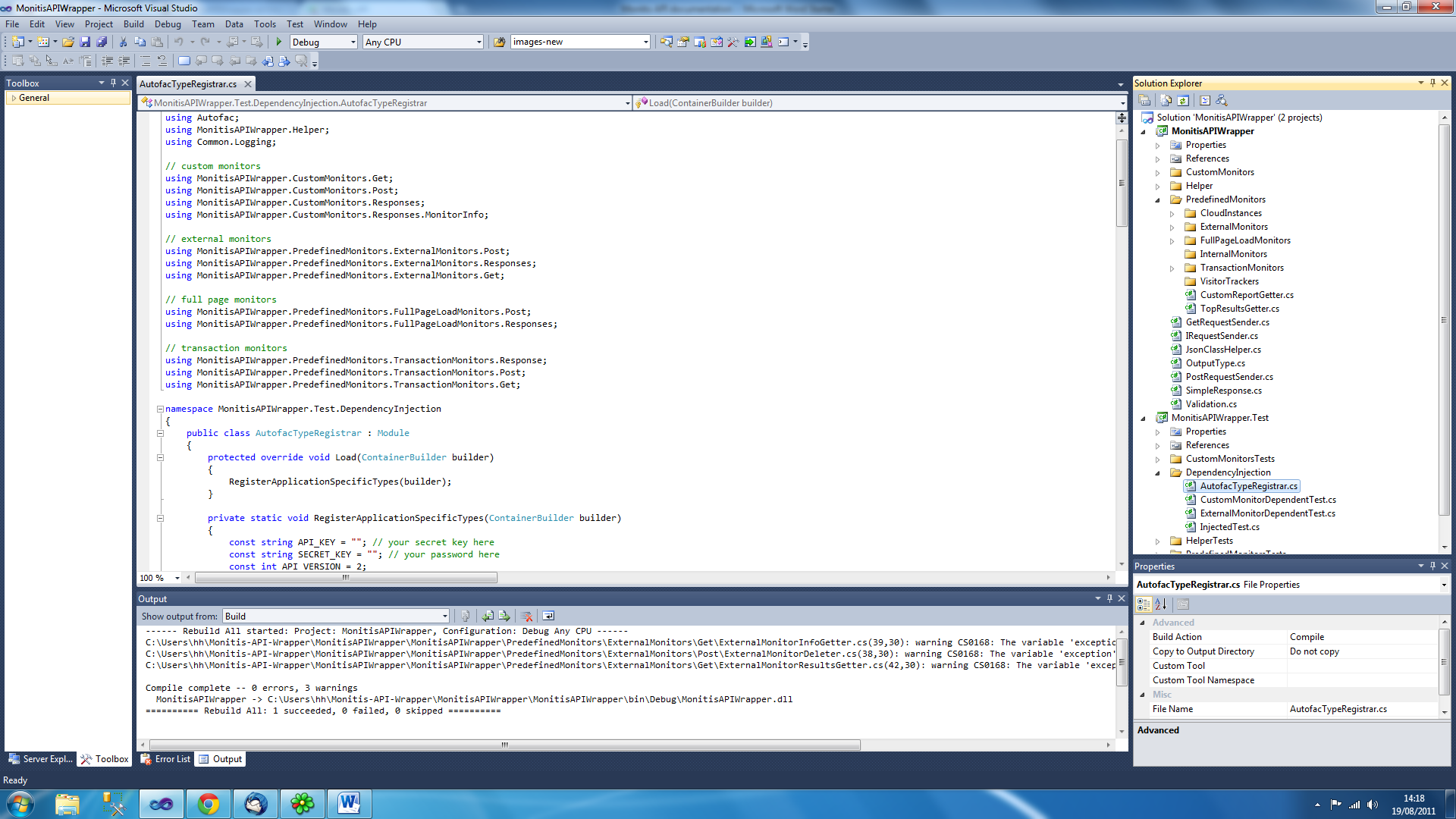
<http://monitis.com/api/api.html>

Any new monitoring classes or methods developed for this project should be done so with the documentation in mind.

**Using the Project**

**Structure**

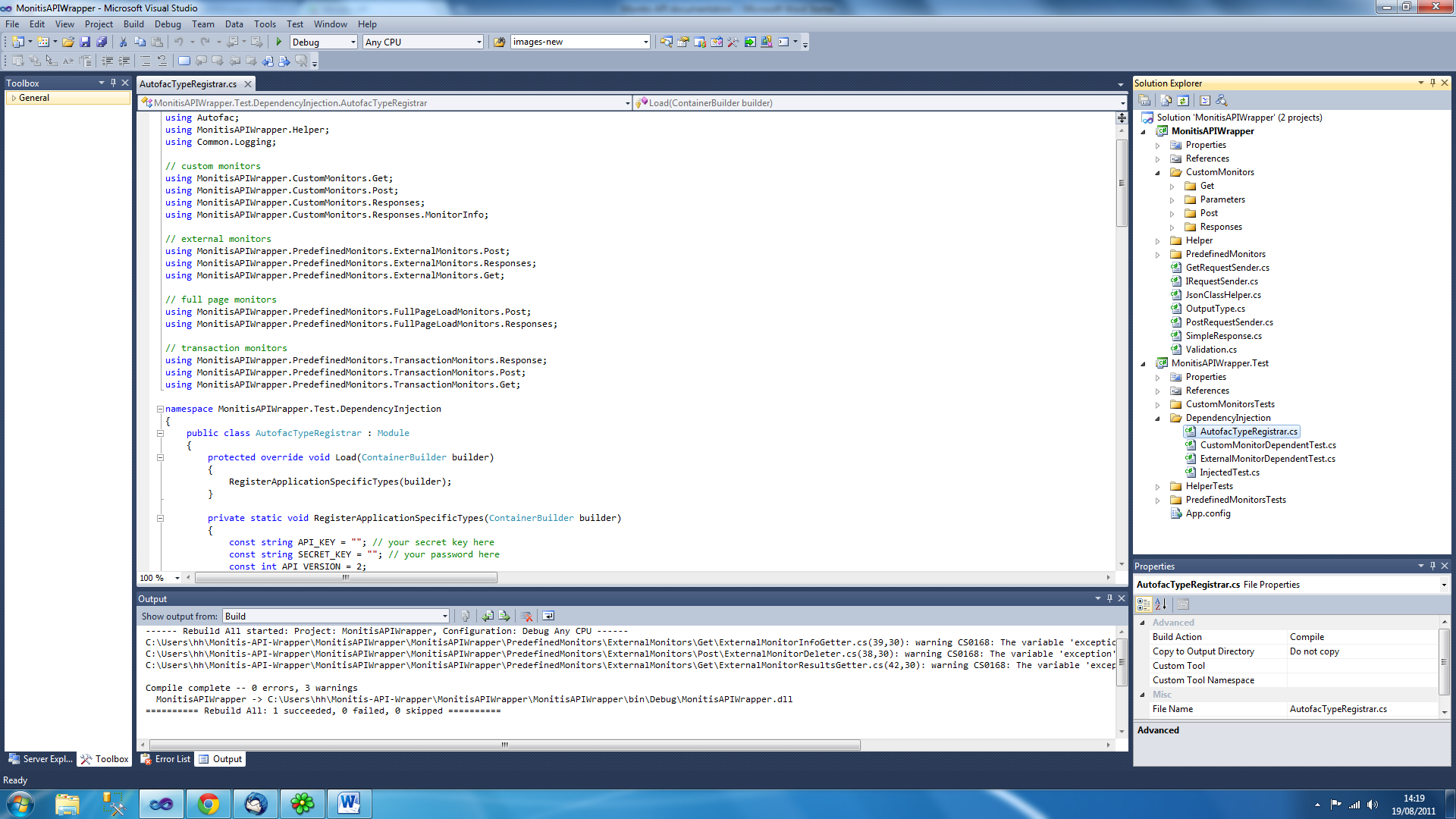
The project folder structure has been designed to match the structure of the request and command classes on the API website. For example, classes used for custom monitor requests are all inside a single folder called CustomMonitors, transaction monitor request classes are inside the TransactionMonitors folder etc.



**Monitor Classes Sub Folders**

Each monitors class folder has a number of sub-folders used for different purposes. These are:

* Get – All monitor classes for this folder that make a request using a GET method
* Parameters – Classes to represent any monitor parameters sent in a request that can only have a fixed number of values such as “Yes” and “No” (see each class for details)
* Post – All monitor classes for this folder that make a request using a POST method
* Responses – All classes used to store responses sent for a monitor class that do fit inside the generic SimpleResponse class. The response class will be named after the monitor class it is used by with Response on the end



**Request Classes**

Every monitor class in the project sends a request using a request class. The request classes are in the main project folder to be used by all current actions or requests as well as any new ones generated by users.

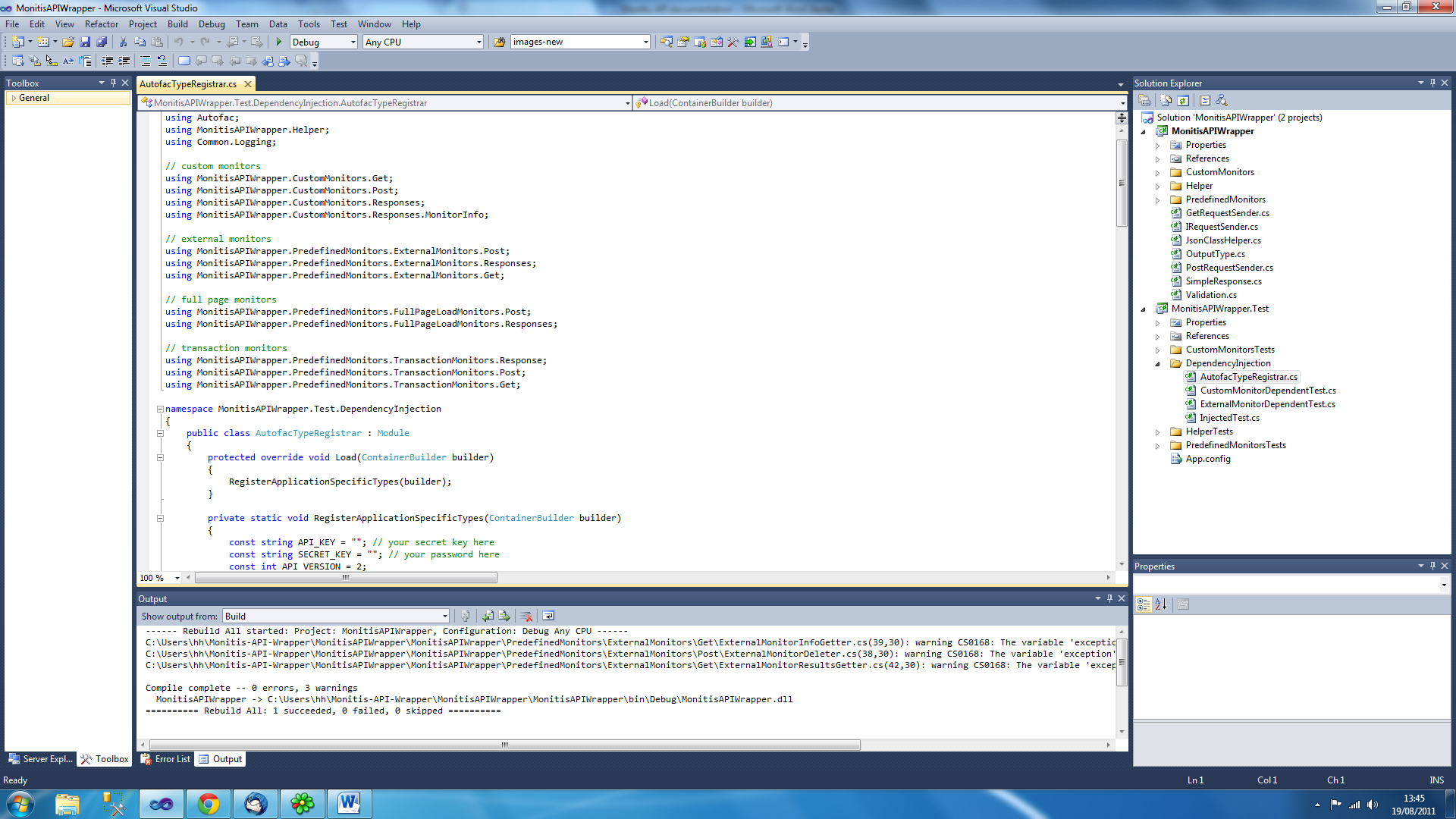
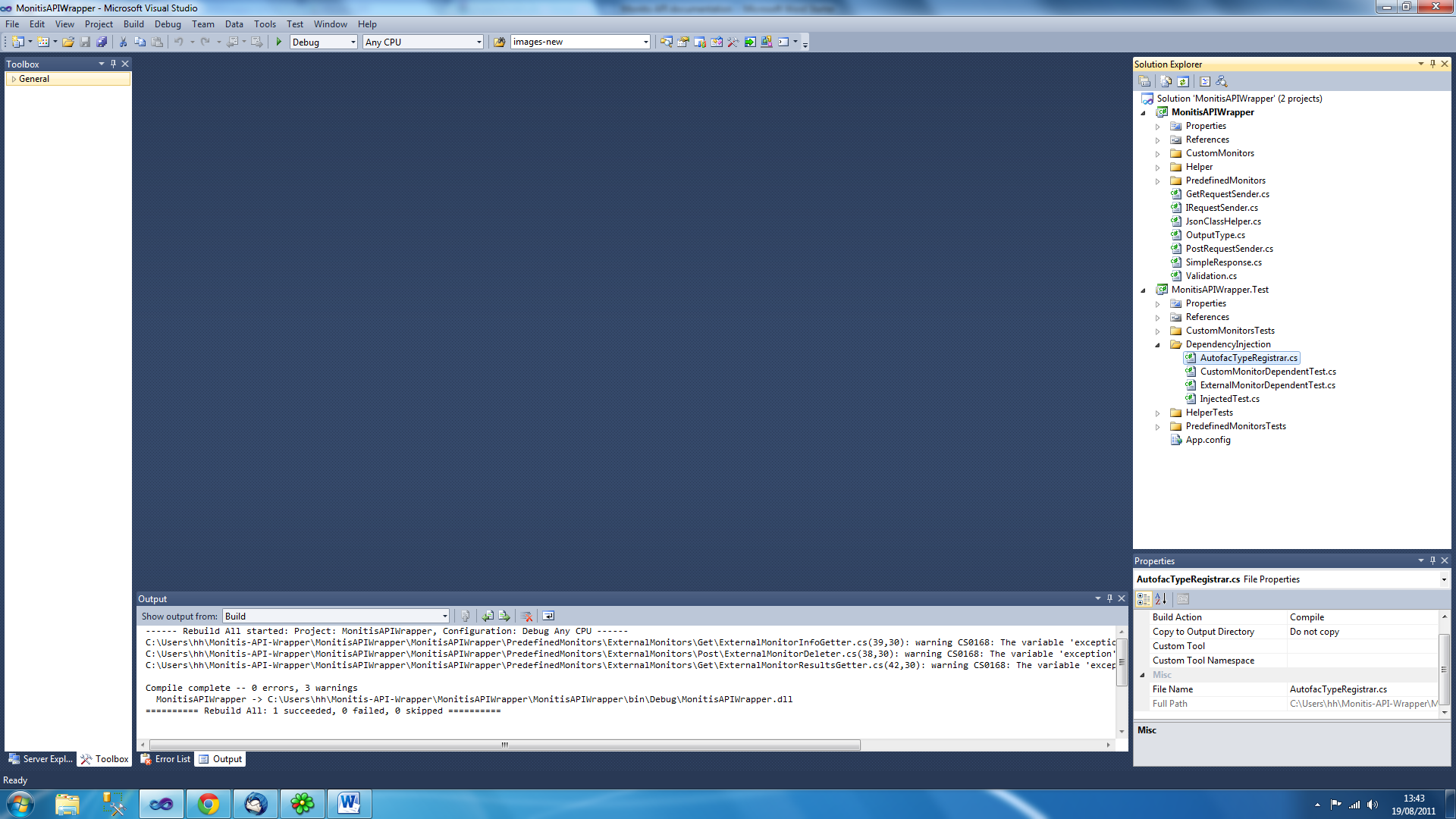
PostRequestSender and GetRequestSender are the two request classes available and the choice depends on whether the monitor class needs to either post a request or get information back.

The IRequestSender interface should to be used in place of these two concrete request classes where possible in code.

**Authentication Details**

An API Key and Secret Key will always need to be provided for any request made.

If you are using and testing the project in the same manner described in this guide, then the API Key and Secret Key will need to be provided as two constant values in the AutofacTypeRegister file. This is in the DependencyInjection folder which is found inside the Monitis API Wrapper Test project.



If you are not planning to use injected values, then you will need to ensure that the API Key and Secret Key values are provided to any instance of the PostRequestSender or GetRequestSender classes you are using before either can make a request.

**Extending**

There are many different types of website monitoring actions and information requests available from Monitis and the project in its current state has only covered the following monitor classes:

* Custom Monitors
* Predefined Monitors -> External Monitors
* Predefined Monitors -> Full Page Load Monitors
* Predefined Monitors -> Internal Monitors

Any others required will need to be created and tested in the project.

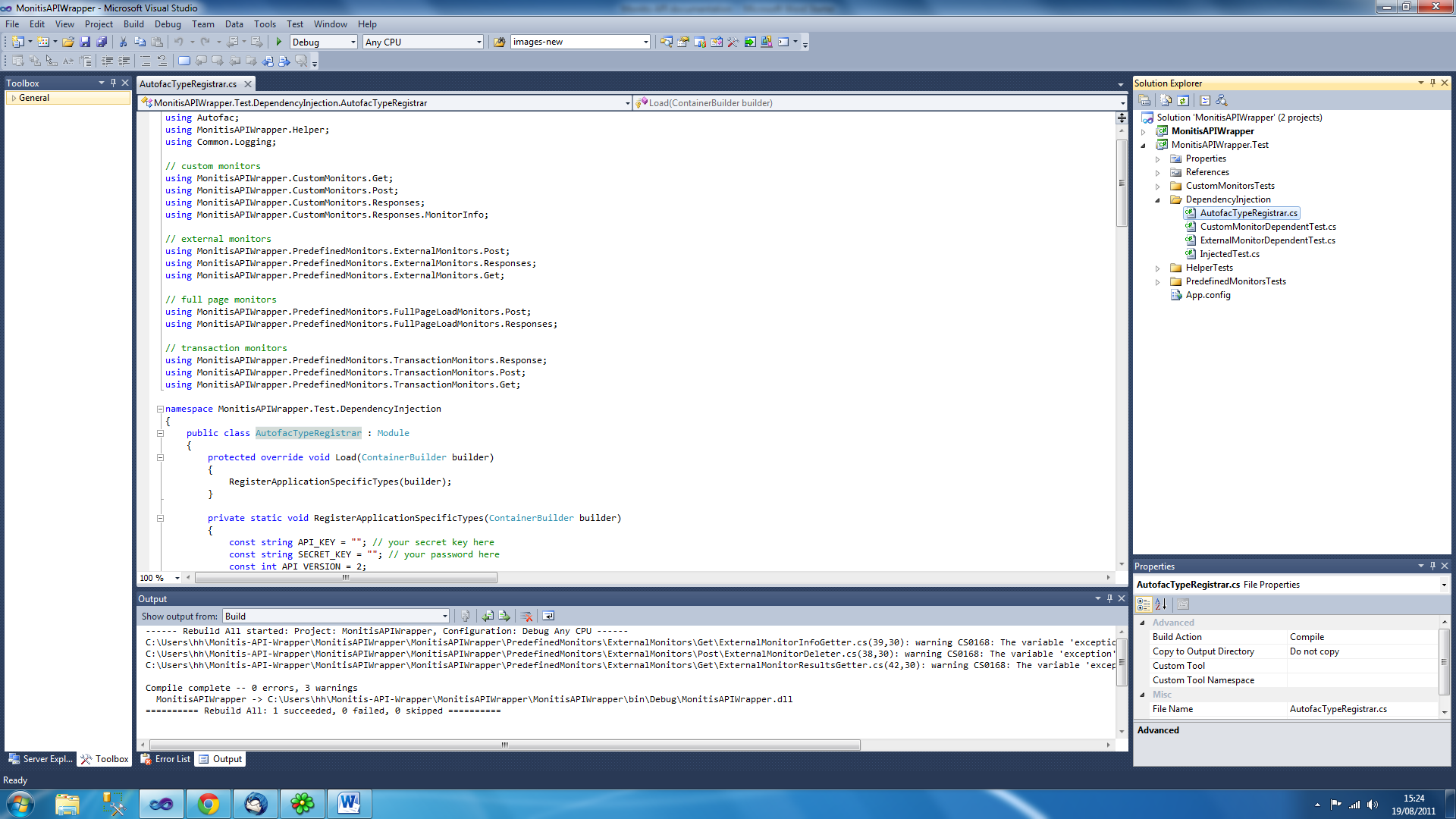
**Testing**

A test project has been included alongside the main project in the solution and tests have been written for every single Get or Post request that can be sent to the Montis website using the project.

Users can extend these tests further if they wish by writing additional tests for the operations available or by making the existing tests more rigorous in some way as well as writing new tests to ensure any new requests that are created are working as intended.

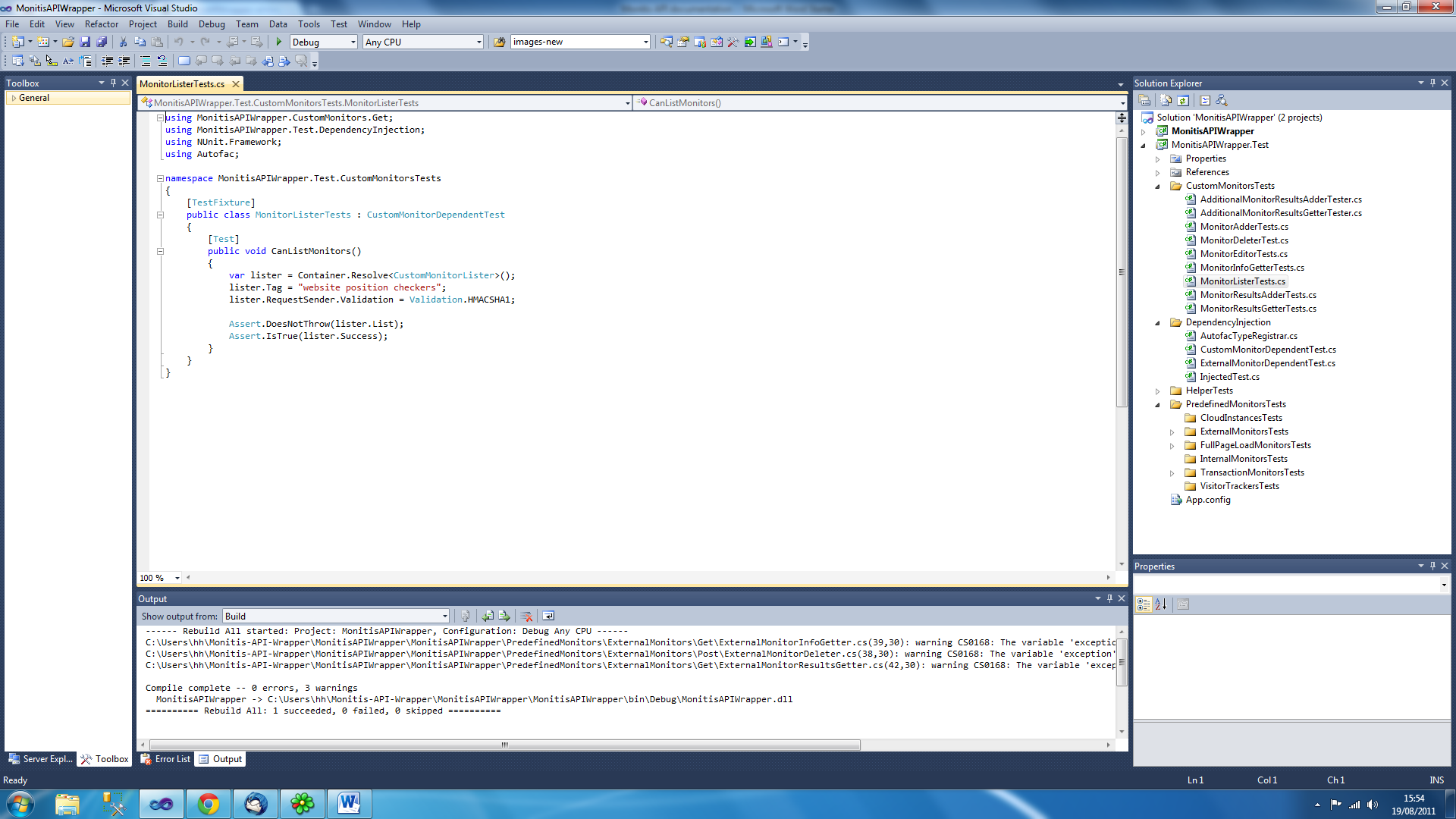
**Injected Values**

The test methods work using injected values where an instance of the monitor class being tested as well as any other classes required (such as a request class) are initialized elsewhere with all values required in a separate class. This class is called AutofacTypeRegistrar.



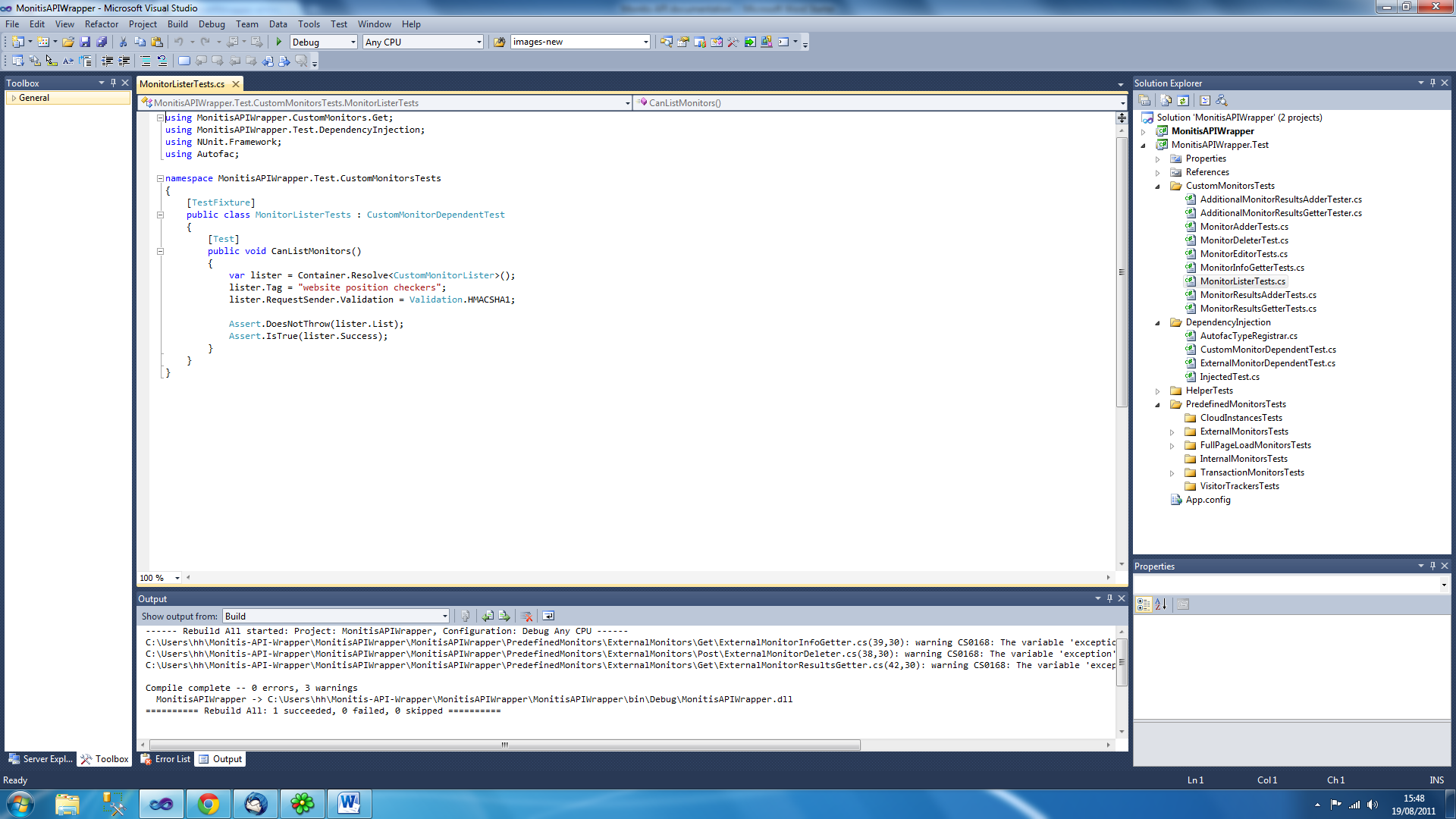
This allows you to then get an instantiated object of the monitor class required, provided you have written a reference for it (and the other classes that it will require) in the AutofacTypeRegistrar class.

Every test class that makes use of injected values will need to inherit from the InjectedTest class (or from a custom class that also inherits from InjectedTest) and also have the following classes in using section.



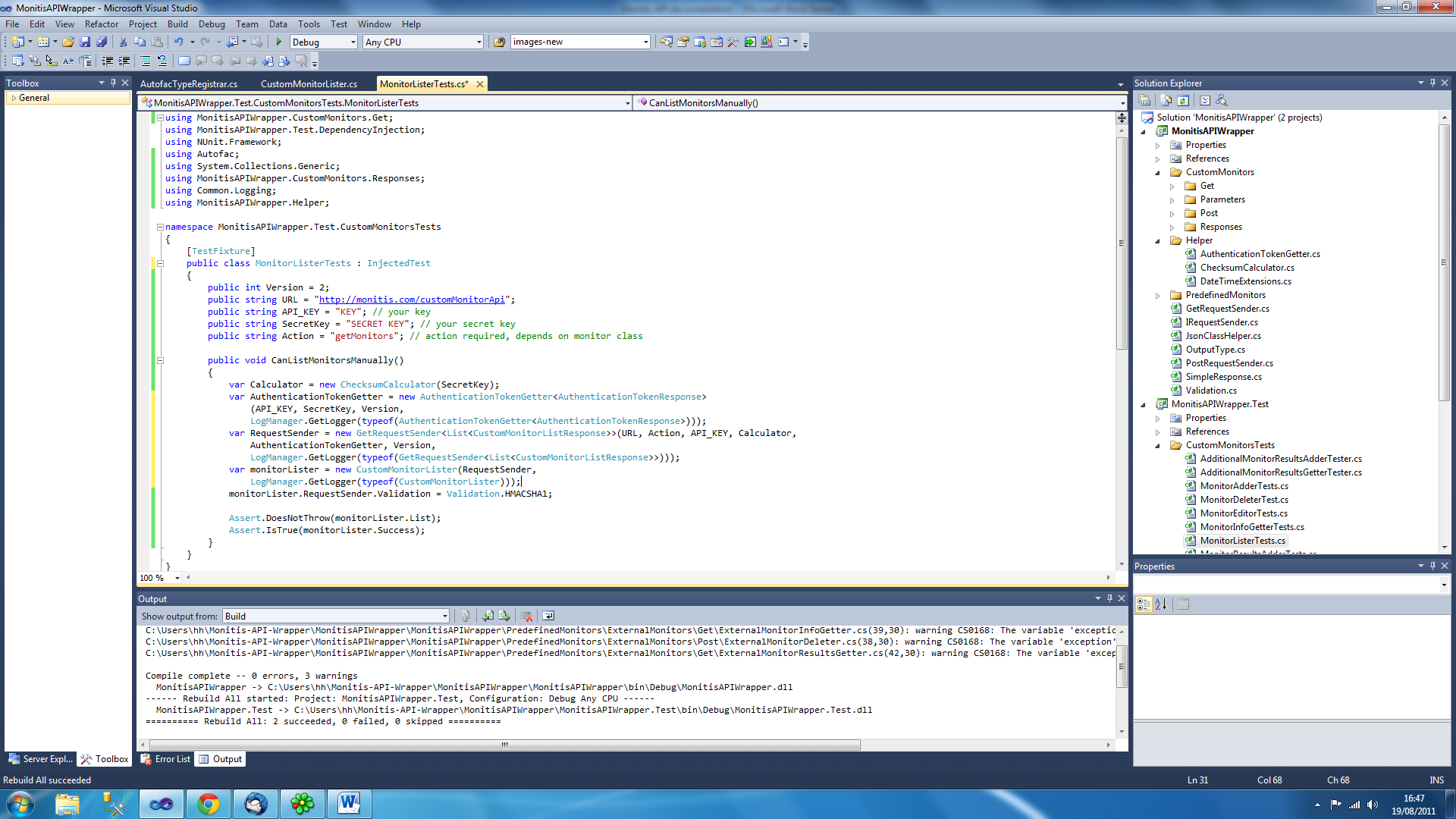
**Test Example**

An example of a test class that makes use of injected classes and values is displayed below. In this example a custom class is used as a parent class instead of InjectedTest.



The Container variable is used to obtain an instance of the class required, in this case a CustomMonitorLister class.

Below is an example of how to perform a test for a monitor if injected properties were not used



Code continues below…

