MAQS Architecture

Prepared By: Troy Walsh

Contents

[Revision History 4](#_Toc41556869)

[Introduction 5](#_Toc41556870)

[Purpose of the Document 5](#_Toc41556871)

[Scope 5](#_Toc41556872)

[Intended Audience 5](#_Toc41556873)

[Development Overview 5](#_Toc41556874)

[Development team 5](#_Toc41556875)

[Process 5](#_Toc41556876)

[Test Entities 5](#_Toc41556877)

[Unit Tests 5](#_Toc41556878)

[Coding standards 5](#_Toc41556879)

[Architectural Design 6](#_Toc41556880)

[Objective 6](#_Toc41556881)

[Modular 6](#_Toc41556882)

[Quick Start 6](#_Toc41556883)

[CI/CD/CT 6](#_Toc41556884)

[Scalability 6](#_Toc41556885)

[Extensibility 6](#_Toc41556886)

[Ease of Use 7](#_Toc41556887)

[Open 7](#_Toc41556888)

[Modules 8](#_Toc41556889)

[Selenium 13](#_Toc41556890)

[Appium 16](#_Toc41556891)

[Database 17](#_Toc41556892)

[Web Services 19](#_Toc41556893)

[Email 21](#_Toc41556894)

[Test Adapters 23](#_Toc41556895)

[Microsoft Unit Test 23](#_Toc41556896)

[NUnit 23](#_Toc41556897)

[Distribution 23](#_Toc41556898)

[Templates 23](#_Toc41556899)

[Binaries 23](#_Toc41556900)

[Documentation 23](#_Toc41556901)

[Wiki 23](#_Toc41556902)

[Source Code 24](#_Toc41556903)

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Version | Description | Revised By |
| 4/11/2016 | Draft | Initial Draft | Jacob Ferm |
| 4/30/2016 | 1.0 | Add version 3 content | Jacob Ferm |
| 3/3/2017 | 1.1 | Rework images | Troy Walsh |
| 4/16/2018 | 1.2 | Update for version 4 | Troy Walsh |
| 8/1/2019 | 1.3 | Update for version 5 | Troy Walsh |
| 8/21/2019 | 1.4 | Update open source links | Troy Walsh |
| 5/27/2020 | 1.5 | Update for version 6 | Troy Walsh |
| 1/5/2022 | 2.0 | Update for version 8 | Troy Walsh |

# Introduction

## Purpose of the Document

This document will cover the architectural design of the MAQS (Modular Automation Quick-Start) framework and its overall implementation.

## Scope

The intent of this document is to provide a comprehensive architectural overview of the MAQS, using different architectural views to depict different aspects of the framework. It will be used to convey decisions which have been made that influenced tools developed and integrated into the framework.

## Intended Audience

This document is intended for teams interested in the design of MAQS.

# Development Overview

## Development team

MAQS is developed by Cognizant Softvision’s QE (Quality Engineering) practice. The people that use MAQS day to day are the same people that build and maintain it.

## Process

The MAQS project is hosted in [GitHub](https://github.com/CognizantOpenSource/maqs-dotnet). Its development follows an agile approach. It also has an automated CI/CD pipeline that allows the team to quickly release new versions.

## Test Entities

Cognizant Softvision maintains multiple resources which allow use to test MAQS. These test entities are what Cognizant Softvision leverages for unit testing the framework. These entities are largely managed in Azure DevOps along with the MAQS project CI/CD resources. The test entities include multiple websites, a web services, a database and an email account.

## Unit Tests

For each project has associated unit test projects. This ensures development changes haven’t broken any MAQS features. No code changes make it into MAQS unless they pass all unit tests.

## Coding standards

MAQS utilizes [SonarCloud](https://sonarcloud.io/dashboard?id=MAQS) to enforce style and consistency rules for each project.

# Architectural Design

MAQS is a quick start comprised of templates and libraries. It is distributed via a Visual Studio extension and NuGet packages. The sections below will provide insight into the design and implementation.

## Objective

Provide a modular testing framework leveraging proven technologies that allows users to start creating meaningful automation fast.

### Modular

MAQS provides several prebuilt modules. There are modules for web, mobile, web services, database and email testing. They are also built on a base module that users are free to extend. This means MAQS allows you to use what modules work for you and if MAQS doesn’t have a module you need you can extend the base module and add your own.

### Quick Start

There is a lot of work that is needed done before you can start creating useful automation. MAQS was designed to take care of a lot of this work. The MAQS libraries provide common functionality, like logging, that would need to be implemented for each automation effort. This allows the engineers to focus on the specific needs of their projects and not the general functionality you need practically every project.

### CI/CD/CT

MAQS was designed with continuous integration, deployment, delivery and testing in mind. It is intended to seamlessly integrate with any modern release pipeline. All modern build servers such as Jenkins, Team City, TFS, Azure DevOps, etc. that have access to a form of C# compiler or runner, MSBuild, MSTest, NUnit Console, allows for execution of automated tests to be included in a continuous fashion.

### Scalability

The framework was built with parallelization in mind. The MAQS team has developed the framework to support running tests in parallel with scalability being dependent on the number of cores available to the machine and the test adapter selected.

### Extensibility

A core tenet of MAQS is that it should not limit its users. This primarily means 4 things. The first is transparency. The MAQS NuGet package contains the source code. If a user what to see how MAQS is doing something they can. The second is that we do not prevent them from getting to the system access objects. The Selenium web driver and web service wrapper are couple of examples of system access objects. The third is that we allow users to override the creation of the system access objects. And finally, MAQS provides a base generic test which engineers can leverage to add their own system access objects.

### Ease of Use

The MAQS solution and class templates can be used as a base to quickly start writing automated tests. The templates include clear commenting and a structure that is easy to understand, allowing new users to understand the template projects and start creating their own tests.

These templates heavily leverage modeling. For Appium and Selenium this means the [page object model](https://www.selenium.dev/documentation/#page-object-design-pattern) and for web service this means data contracts. The modeling strategies help reduce maintenance, reduce duplicated code and it makes your test code cleaner and more concise.

### Open

MAQS heavily relies on open source tools. The type of tools we have chosen provides 4 primary benefits. First, it means you don’t need to pay for any licenses to use the software. Second, you can get updates on your own schedule, not on a vendor’s. Third, you have the power and right to add new capabilities. And lastly, if something breaks you have the power to fix it yourself.

Here is the list of licenses for MAQS and the technologies it depends on:

* MAQS: <https://github.com/CognizantOpenSource/maqs-dotnet/blob/main/LICENSE> - MIT
* Selenium: <http://www.apache.org/licenses/LICENSE-2.0> - Apache 2
* Selenium.Axe: <https://raw.githubusercontent.com/TroyWalshProf/SeleniumAxeDotnet/master/LICENSE> - MIT
* Appium: <https://raw.githubusercontent.com/appium/appium-dotnet-driver/master/LICENSE> - Apache 2
* Dapper: [https://raw.githubusercontent.com/StackExchange/Dapper/master/License.txt](https://raw.githubusercontent.com/StackExchange/Dapper/master/License.txt%20) - Apache 2
* MongoDB: <https://www.nuget.org/packages/MongoDB.Bson/2.10.4/License> - Apache 2
* Microsoft ASP.NET Web API Client <https://github.com/aspnet/AspNetWebStack/blob/master/LICENSE.txt> - Apache 2
* NUnit: <http://nunit.org/nuget/nunit3-license.txt> - MIT
* MSTest V2: <https://github.com/microsoft/testfx/blob/master/LICENSE.txt> - MIT
* Dotnet Extensions: <https://github.com/dotnet/extensions/blob/master/LICENSE.txt> - MIT
* Dotnet Runtime: <https://github.com/dotnet/runtime/blob/master/LICENSE.TXT> - MIT
* Dotnet Standard: <https://github.com/dotnet/standard/blob/master/LICENSE.TXT> - MIT
* Json.NET: <https://raw.githubusercontent.com/JamesNK/Newtonsoft.Json/master/LICENSE.md> - MIT
* MailKit: <https://github.com/jstedfast/MailKit/blob/master/License.md> - MIT
* WebDriverManager.Net: <https://github.com/rosolko/WebDriverManager.Net/blob/master/LICENSE> - MIT
* Cucumber: <https://github.com/cucumber/common/blob/main/LICENSE> - MIT
* SourceLink: <https://github.com/dotnet/sourcelink/blob/main/License.txt> - MIT

## Modules

This section will review each type of modules inside MAQS. Below is a diagram showing the high-level architecture of the framework.

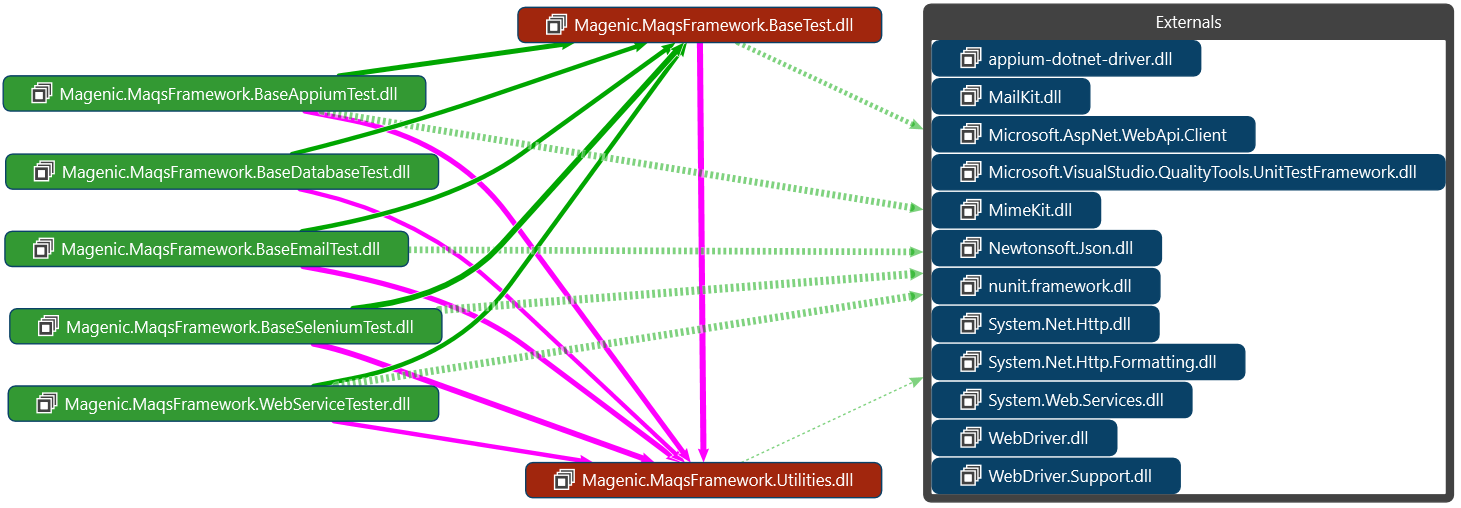


Figure 1. Diagram of MAQS Framework

#### Base Project

The base project consists of a mindset to leverage both NUnit and Visual Studio Unit Test libraries to tailor written tests to the project requirements. MAQS is developed to be modular, allowing testers to only import the parts of MAQS required for the project. This reduces the clutter involved with importing tools into a project and allows for integrating each type of project together for composite testing

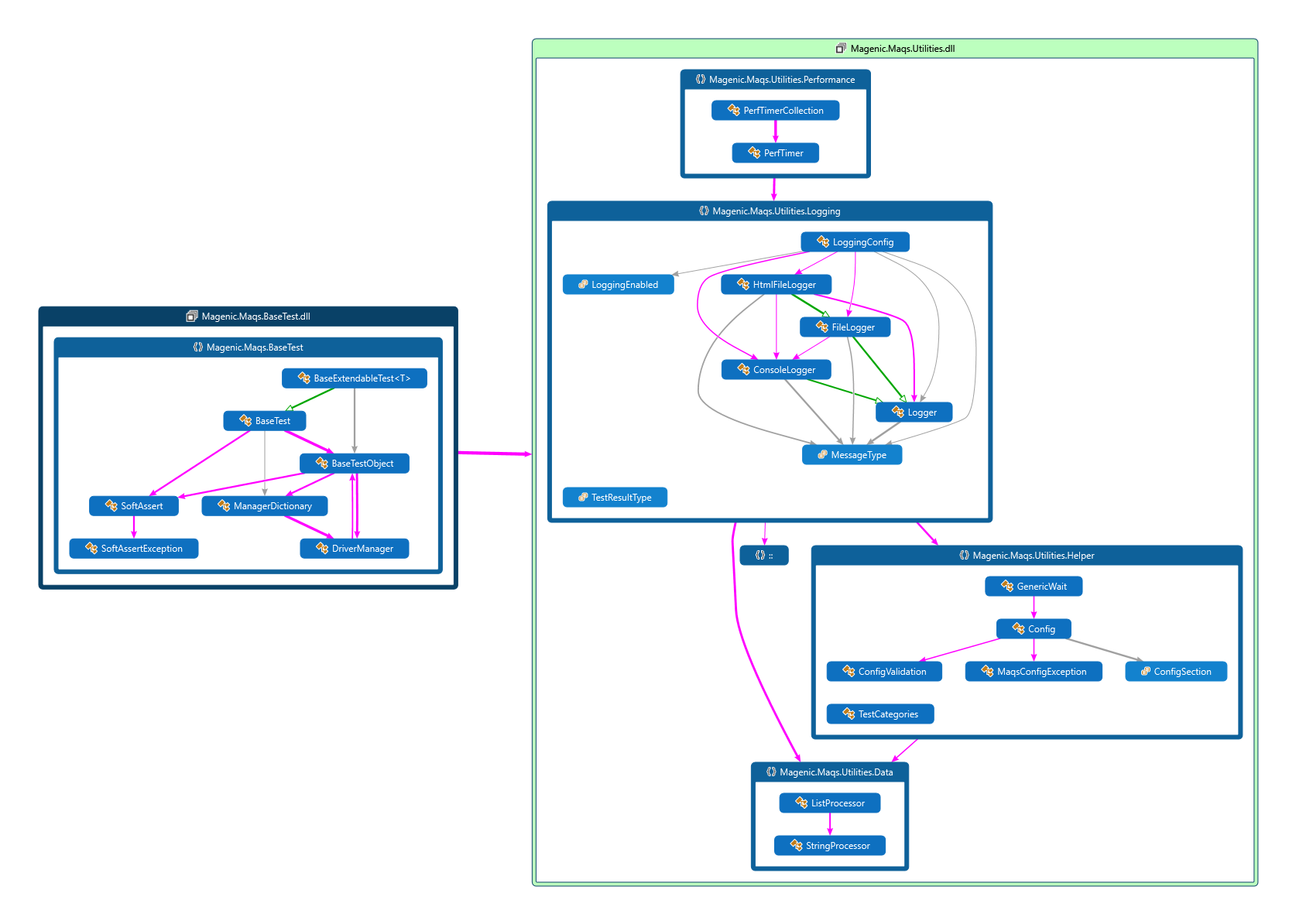


Figure 2. Diagram of BaseTest and Utilities

Tests utilize the page model to write automated tests. Tests are created to maximize readability; tests are simple to read even for those without or limited development background. With proper construction, it reads like a keyword driven framework without the drawback of a keyword driven frameworks lack of complexity.

#### TestObject

Each test that gets executed gets its own test object. Test objects are a thread safe. The test object holds the system access object, logger, manager store collection, soft assert collection, performance timers, a dictionary of string values as well as a dictionary of objects. Test objects are a key part of how MAQS tests can run in parallel.

#### Generic Wait

MAQS has built in generic wait functions that can be used to set different varieties of waits for a project. The waits included in the framework are functions to return true, waiting for expected return types, and passing arguments. Rules for timeout and retry interval are set inside the wait functions. Different rules for timeouts and exceptions inside waits provide the engineer extensive options when writing tests.

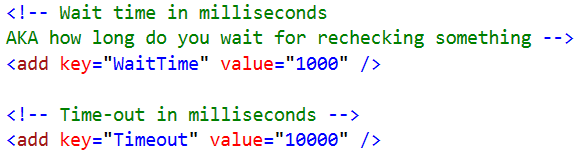


Figure 3. Example of wait configurations

#### Soft Asserts

Soft asserts allow for automated tests to continue even if an asserted condition fails. This allows for multiple validations to be performed during test execution before failing the test if one or more asserts fail. Finishing a test ensures that all the steps are completed before failing.

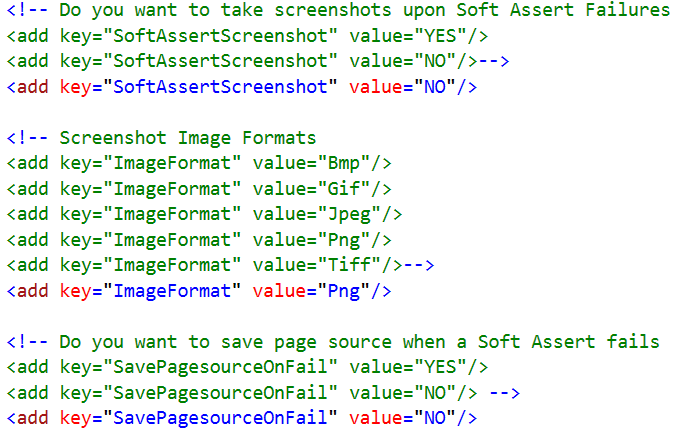


Figure 4. Example of soft assert configurations

#### Logging

There are three logging types included with the base project: TXT, RTF, and Console. These logging types can be set inside the configuration file for the desired project along with the desired location to store the files. Another property is the enablement of the logging. Included configurations are enabling the logger to create a log on test failure, always output, or never output a log.



Figure 5. Example of logging configurations

#### Response Time

The response time feature allows for accurate timing during a test to measure the speed at which actions are performed. Solutions can be written to test web service requests, the speed at which a browser completes loading an item on a page, and response speeds, or the time it takes to retrieve a query from a database. Response times can be used to give a snapshot of the performance of a system.

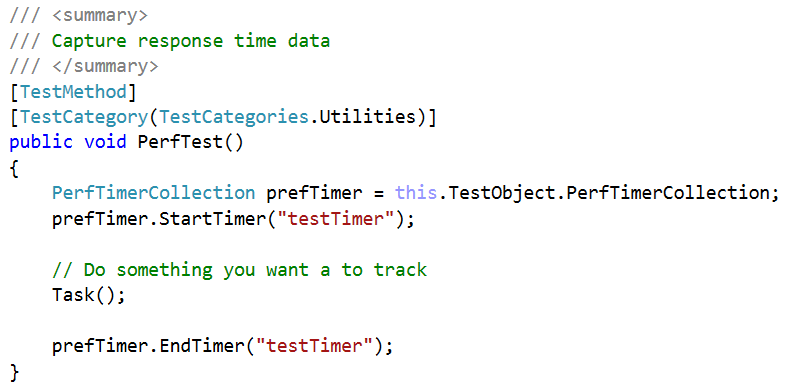


Figure 6. Example of using a response time logger

After the test has completed, the results are logged to an XML file to be consumed for desired project needs.

#### Manager Store

Drivers are a core concept within MAQS. A driver is an object used to interact with a system under test. A couple of examples are the Selenium web driver and the Database driver. Drivers are stored in driver manages. These managers are responsible for lazily setting up and cleaning up after drivers. The Manager Store is essentially a collection of these managers.

This design has several advantages. First the underlying drivers are not initialized until they are needed. Second this allows us to provide different initialization functionality for different drivers. Finally, it allows you to use and interact with multiple systems under test within a single test. Having multiple systems under test isn’t common, but it is very nice to have when those rear cases where it is needed.



Figure 7. Example of using the manger store

It however does more than just stores drivers

The manager store is a collection of system under test drivers.

### Selenium

[Selenium](http://www.seleniumhq.org/) is the most frequently used open-source web automation tool in test automation. Selenium tests are run against web applications, using Selenium WebDriver to drive navigation and interactions.

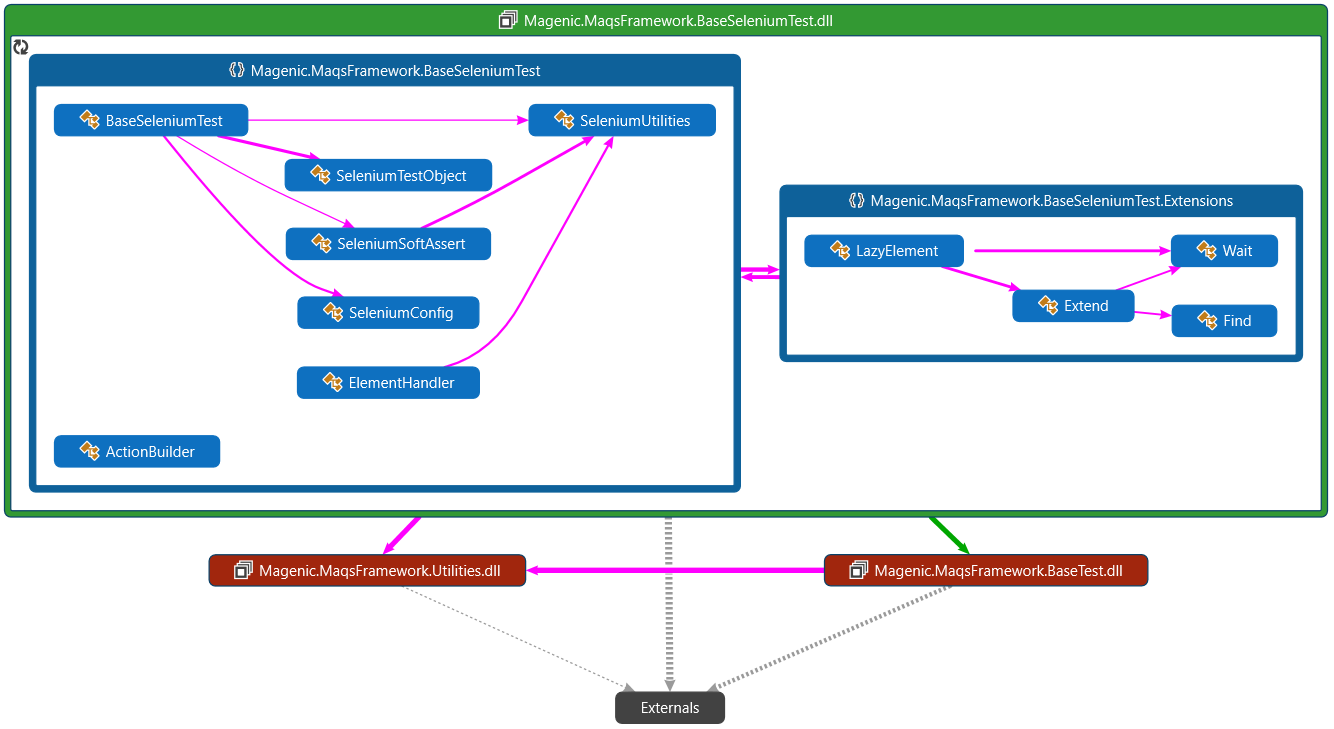


Figure 8. Diagram of Base Selenium Test

#### Selenium WebDriver

Selenium WebDriver utilizes a browser-specific driver to send commands to a browser and retrieve results. The WebDriver launches and runs instanced browsers that are configured by MAQS.

#### Selenium Grid

Selenium Grid is a server that allows tests to use web browser instances running on remote machines. The ability to run tests on remote browser instances is useful for spreading testing load across multiple systems. MAQS is designed to easily integrate with cloud bases grid services such as [Sauce Labs](https://saucelabs.com/).

#### WebDriver Waits

MAQS has custom “Wait For” and “Wait Until” functions that eliminate the need to use the system sleep. Writing an explicit sleep for 10 seconds will have the test wait for 10 seconds before proceeding to the next step. WebDriver waits cuts down on testing time by continuing to test an expected condition over a set interval retry. When the expected condition has been satisfied, the test continues. If the timeout has been occurred, the test will fail. These waits can be utilized for responsive UI web applications that lazy load data in the background.

#### Lazy Element

Selenium is often criticized for being flakey. This is largely due to the asynchronous nature of modern web sites. Users have adopted many strategies for addressing this such simply rerunning failed tests and/or adding hardcoded sleeps. These strategies often bloat execution time and/or hide real bugs.

MAQS addresses the inherent flakiness of Selenium with something called Lazy Element. Lazy Element implements the Selenium web element interface so it can be used like any other web element. Lazy Element differs from traditional web elements in three important ways. First, the lower lever web element finds and interactions are wrapped by web driver wait and generic wait functionality. What this does is allows the automation to dynamically wait for an element to exist and/or be in the expected state before interacting with it. For example, if you execute the click command on an element that isn’t clickable yet, MAQS will wait (up to our configured timeout) for the element to become clickable before preforming the click. Second, the find is executed in a lazy manor. This means MAQS waits for the code to try an interact with the element before executing the element find. MAQS also caches the element so we don’t need to find it again later. Third, it provides better/more readable event logging than the event firing web driver.

#### Selenium Configurations

An application configuration (App.config or appsettings.json) file is included in the framework templates. There are structured settings with pre-written configurations for running tests in different browsers or in parallel over a hub using Selenium Grid. There are also default settings that can be overrode with the configuration files. A couple examples are wait times, timeouts and logging types.

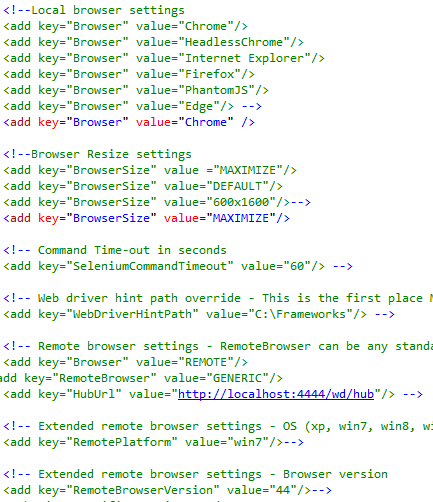


Figure 9. Example of Selenium configuration settings

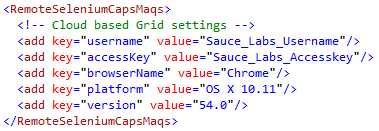


Figure 10. Example of Selenium remote capability configuration settings

#### Screenshots

MAQS includes functionality to capture a screenshot when a test is met with certain conditions, such as a failed test or unexpected error. The automation engineer can use the screenshots to help pinpoint the point of error and diagnose the issue. There are functions available to allow for screenshots to be taken at any point in a test.

### Appium

Appium is an extension of Selenium. It allows you to automate mobile applications on Android and IOS devices. As Appium is an off shoot of Selenium the MAQS Appium capabilities are very similar to those of MAQS Selenium capabilities.

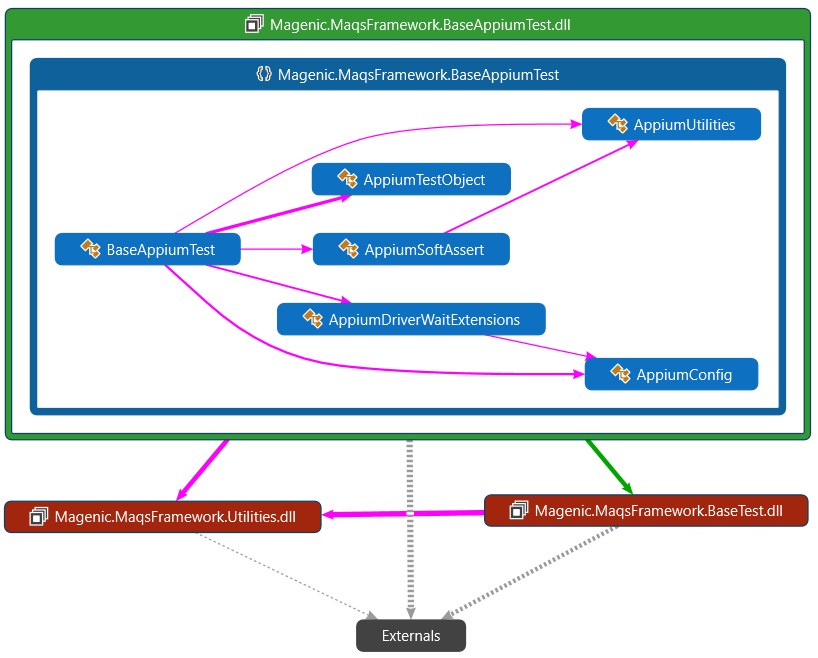


Figure 11. Diagram of Base Appium Test

#### Appium WebDriver

Appium AppiumDriver utilizes a mobile-specific driver to send commands to a mobile device and retrieve results. The AppiumDriver launches and runs instanced mobile application or web application that are configured by MAQS.

#### Selenium Grid

Appium can leverage Selenium grid and the same manner as Selenium. Appium MAQS can also easily integrate with cloud bases grid services such as [Sauce Labs](https://saucelabs.com/).

#### AppiumDriverWaitExtensions Waits

MAQS has custom “Wait For” and “Wait Until” functions that eliminate the need to use the system sleep. Writing an explicit sleep for 10 seconds will have the test wait for 10 seconds before proceeding to the next step. AppiumDriver waits cuts down on testing time by continuing to test an expected condition over a set interval retry. When the expected condition has been satisfied, the test continues. If the timeout has been occurred, the test will fail. These waits can be utilized for responsive UI web applications that lazy load data in the background.

#### Lazy Mobile Element

The lazy mobile element is simple the Appium implementation of the MAQS Selenium [lazy element](#_Lazy_Element). It works the same way and has all the same benefits as the Selenium implementation.

#### Appium Configurations

Appium configurations are included in an app.config (or appsettings.json) file inside the project solution. The file contains configurations for connecting to the mobile device, what mobile application to interact with, rules for time-outs, when to create a log file, and what type of log file is created.



Figure 12. Example of Appium configuration settings

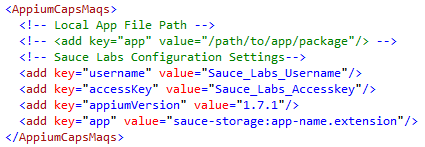


Figure 13. Example of Appium remote capability configuration settings

### Database

MAQS database testing supports running SQL commands against a database. It can run queries, return query results, run stored procedures and log results. In the tests you can specify full SQL queries, define parameters, and specify pass or fail conditions.

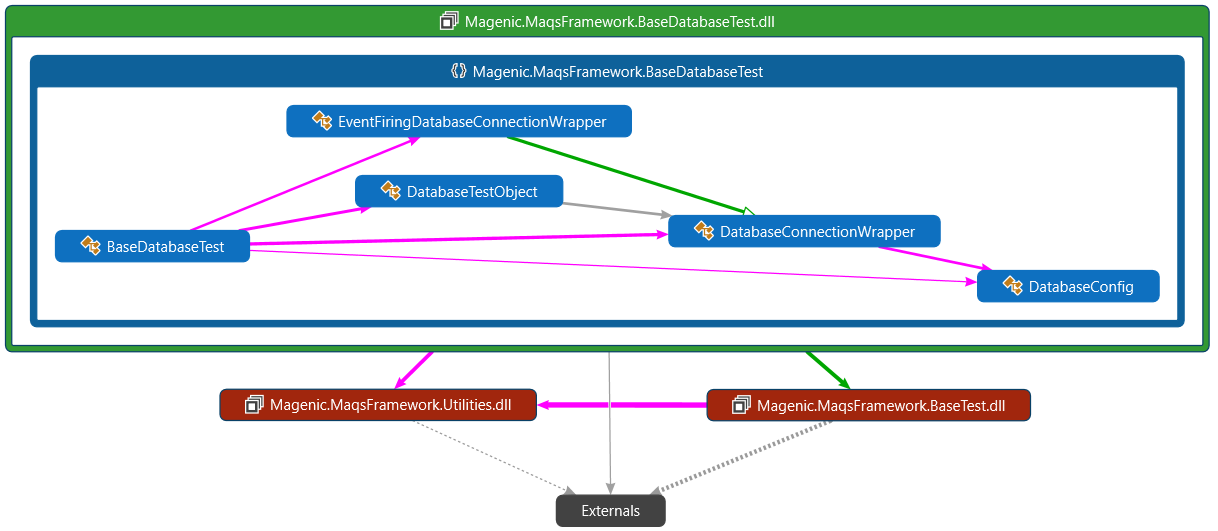
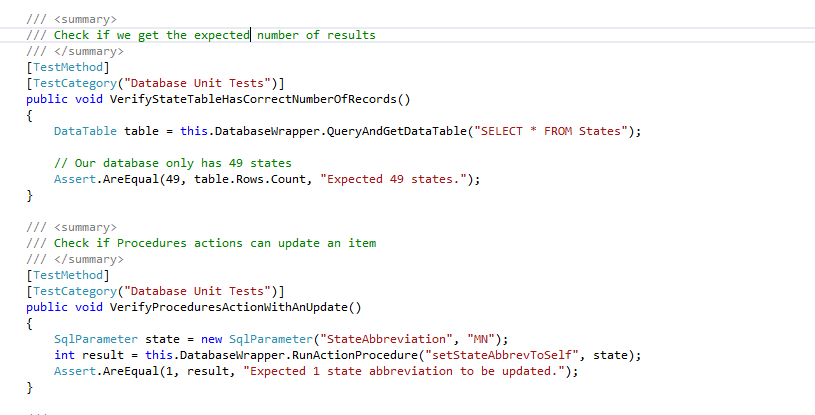


Figure 14. Diagram of Base Database Test

Figure 15. Example of a test that is querying database

#### Database Configuration

Database configurations are included in an app.config file inside the project solution. The file contains configurations for connecting to a database, rules for time-outs, when to create a log file, and what type of log file is created.

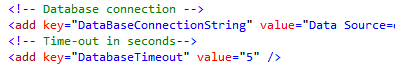


Figure 16. Example of database configuration settings

### Web Services

MAQS web services utilizes the ASP.NET Web API framework. With the framework, it is simple to build web service calls for a variety of functions. MAQS includes functions to expect a successful call, a HTTP response back from a web service, the string content returned, creating HTTP content, as well other functions.

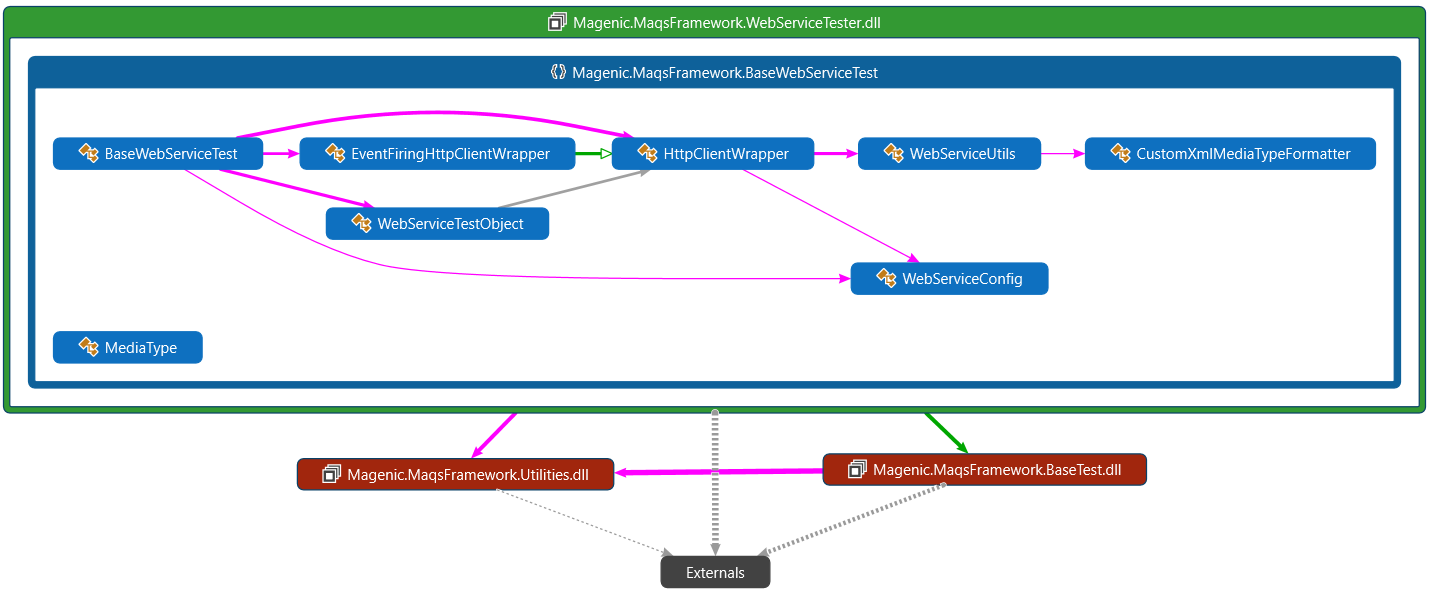


Figure 17. Diagram of Base Web Service Test

#### Web Service Model

Web service models are created for any type of data format. They contain models of the object that will be sent or received through the web service calls.

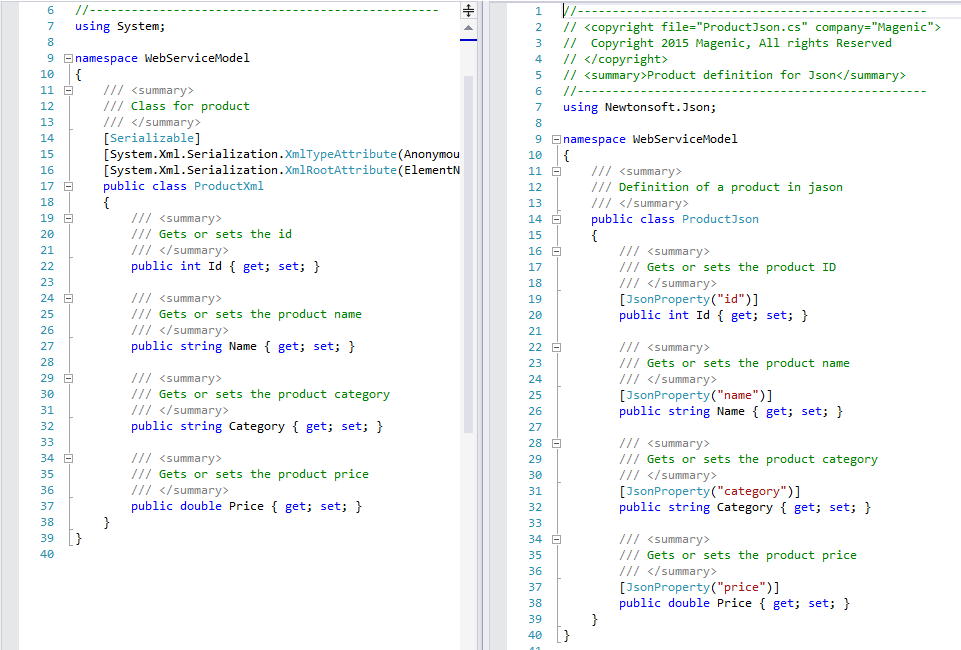


Figure 18. Example of XML and JSON Model

#### Web Service Tests

Web service tests can contain model creation, test details, method calls to the web service, and actions. Acceptance criteria can be written to determine under which conditions web services tests pass or fail.

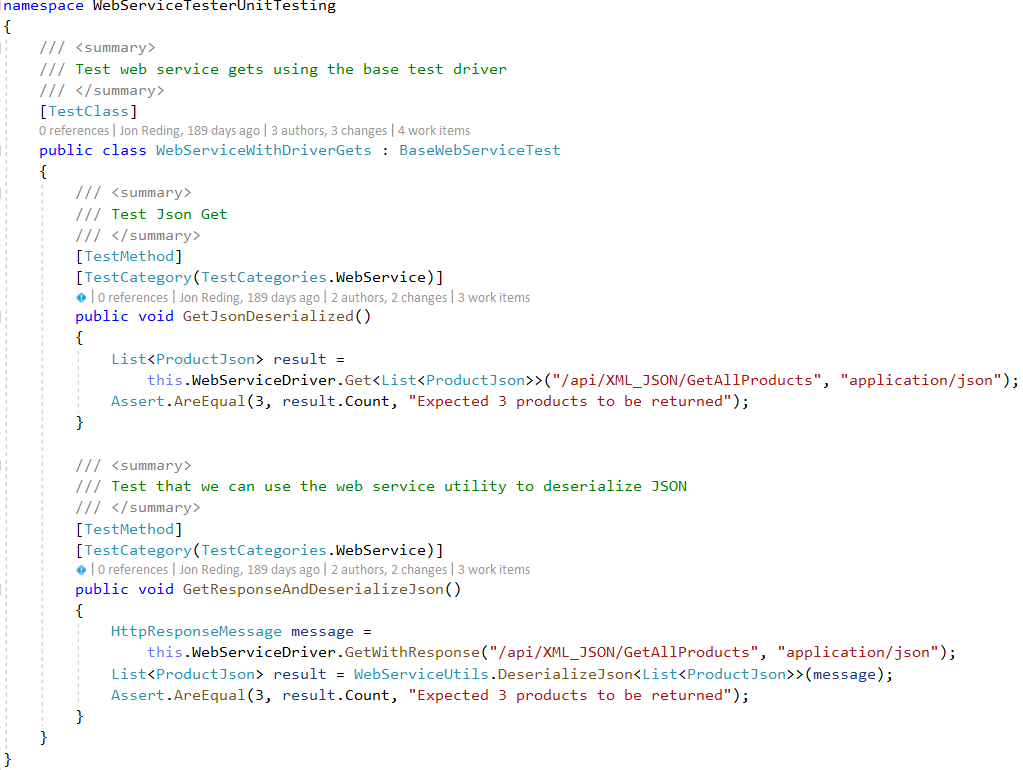


Figure 19. Example test of JSON test

#### Web Service Test Configuration

WebService test configuration settings are included in a configuration file, where settings such as changing WebService timeouts, choosing when a log is created for a test, and what type of log is output.

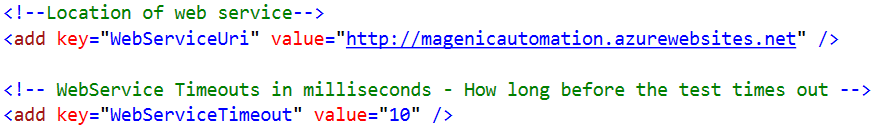


Figure 17. Webservice App.config

### Email

MAQS email testing supports interactions with an IMAP capable email. It can search for, read, download attachments, move and delete emails.

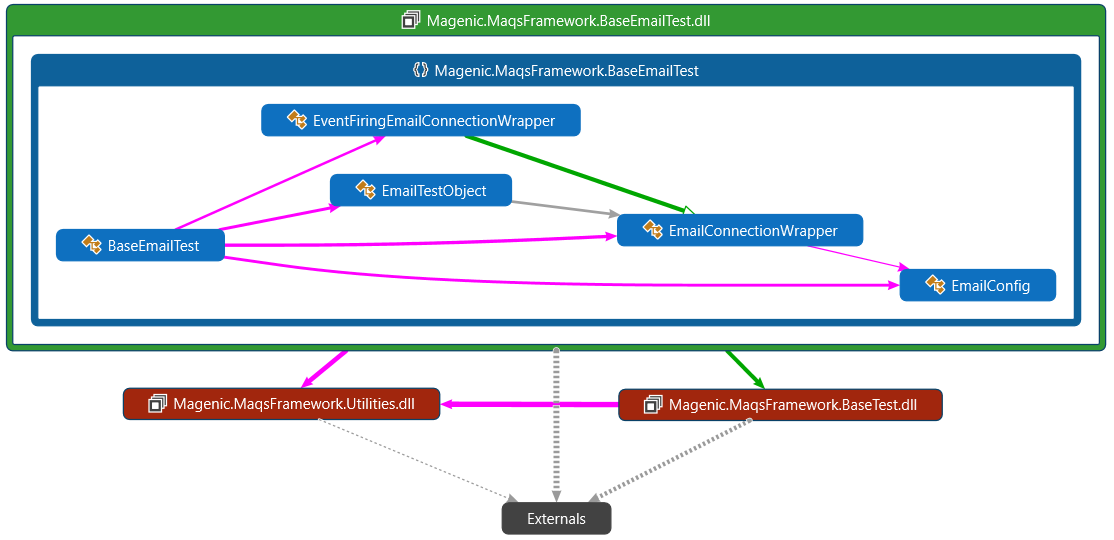


Figure 20. Diagram of Base Email Test

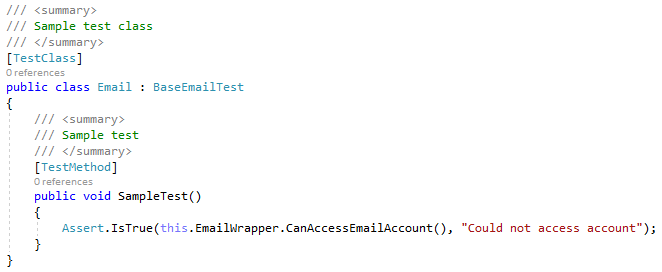


Figure 21. Example Email Test

#### Email Configuration

Email configurations are included in an app.config file inside the project solution. The file contains configurations for connecting to the email server and an optional download override location.



Figure 22. Example of email configuration settings

## Test Adapters

MAQS utilizes the Microsoft unit testing framework and NUnit. Leveraging these unit testing frameworks allows MAQS to easily integrate practically any modern build server. This integration is essential when adding automated testing to your DevOps release pipeline.

### Microsoft Unit Test

The Microsoft Unit Test Adapter is included with Visual Studio; it is a framework that supports unit testing in Visual Studio. Microsoft unit test lets testers check that code is working as expected. It allows tests to be run and viewed inside Visual Studio.

### NUnit

The [NUnit 3.0 test adapter](https://github.com/nunit/docs/wiki/VS-Adapter-Documentation) is an extension for Visual Studio that allows for discovery and execution of unit tests. Once the extension is installed, the test adapter integrates itself inside the Visual Studio Test Explorer to execute tests. NUnit tests can be run just like Visual Studio Unit Tests if the NUnit libraries are used within the automation project.

# Distribution

## Templates

MAQS includes an extension installer for Visual Studio. This [VSIX](https://marketplace.visualstudio.com/items?itemName=vs-publisher-1465771.MAQSOpenFramework) file includes templates for each type of project included in the framework. There are templates included for NUnit and Visual Studio Unit Tests with each type of project: Appium, Database, Email, Selenium tests, Web Services and Composite (mixture of the other types) tests.

## Binaries

MAQS utilizes NuGet to distribute the testing framework. When a new version is ready for release the Cognizant Softvision team starts the release build which triggers the NuGet package builder. Once a new version of MAQS is ready it gets pushed to [NuGet](https://www.nuget.org/packages/Magenic.Maqs/).

## Documentation

This section will outline the documentation provided with the MAQS framework.

### Wiki

All MAQS documentation is hosted in the GitHub [wiki](https://magenic.github.io/MAQS/#/). The wiki is updated as part of the normal course of MAQS development.

### Source Code

All MAQS source code is public and hosted in [GitHub](https://github.com/Magenic/MAQS).