Test plan

# Introduction

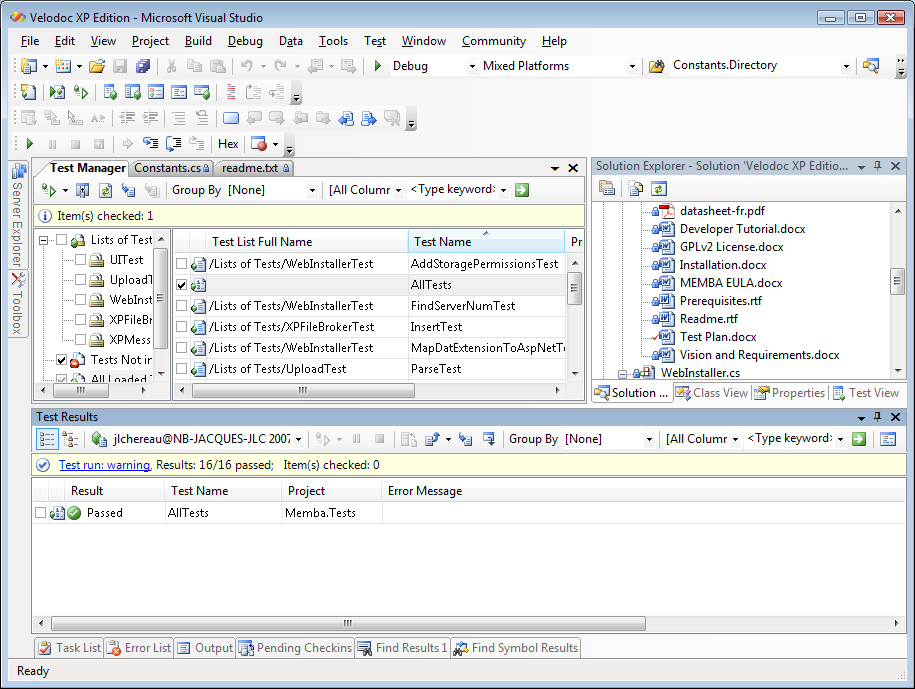
The test plan submitted with this project is kept minimal so as to be easy to reproduce. We have unit tests which rely on Visual Studio Unit Testing framework and simple load tests which use TinyGet.exe.

# Units tests

All unit tests reside in the Memba.Tests project of the Velodoc XP Edition solution.

Before running units tests, you need to configure your test environment:

1. Create an IIS virtual directory which you can play with. You can map the IIS virtual directory to the directory of the web site in the Velodoc XP Edition solution but this is not required. The IIS virtual directory is only used here to test the methods of the WebInstaller class which create a script map for the .dat extension and reassign the Http Error 404 page.
2. Modify the App.config file in Memba.Tests to:
   1. designate the correct storage folder in the connectionString attribute of the add element corresponding to the FSFileStorageProvider;
   2. configure properly the smtp settings for sending emails.
3. Modify the Constants.cs file in Memba.Tests according to the comments to map your configuration.

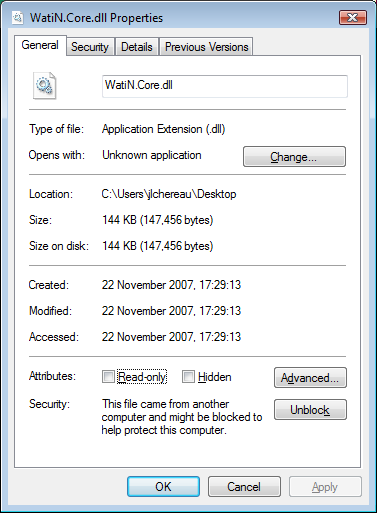


To run the unit tests, simply launch the test manager and click run. We have also provided a list of ordered tests for your convenience as the SelectByGuidTest should follow InsertTest. It may be easier to run the AllTests list.

## Known issues:

#### WatiN assembly may be locked which will prevent its execution

You may get the following message: *Test Run deployment issue: The location of the file or directory <assembly path> is not trusted*. There are good chances this is related to the reference to WatiN.Core.dll. Edit the file properties from windows explorer and if you see *This file came from another computer and might be blocked...*click the Unblock button



#### WCF streaming services do not work in Cassini

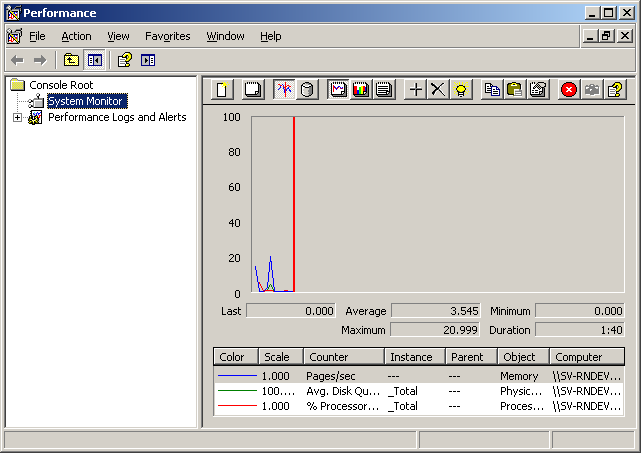
WCF streaming services do not work in Cassini aka. Visual Studio Development Web Server; They only run in IIS.

# Load tests

Visual Studio load testing framework is not capable of submitting a large number of large posts.

There are several solutions which might be considered including using Microsoft Web Capacity Analysis Tool (WCAT). See: <http://msdn2.microsoft.com/en-us/library/ms951774.aspx>. WCAT can be found in the IIS Resource Kit.

WCAT requires some infrastructure to deploy and we have chosen to demonstrate here a very simple solution based on TinyGet which is also part of the [IIS Resource Kit](http://www.microsoft.com/downloads/details.aspx?FamilyID=56FC92EE-A71A-4C73-B628-ADE629C89499). We need to gather some metrics and we use Performance Monitor for this purpose. We recommend you read [Improving .NET Application Performance and Scalability](http://www.microsoft.com/downloads/details.aspx?FamilyId=8A2E454D-F30E-4E72-B531-75384A0F1C47) which will give you loads of details about the counters to include. I will not detail this part of the load test session.



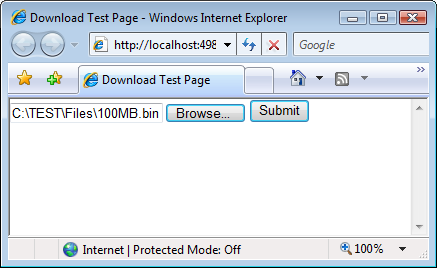
First you need to configure the RequestStreamDumpModule in the <configuration><system.web><httpModules> section of web.config. This module should replace the UploadHttpModule which needs to be removed. The best is to comment it as it will need to be restored.

<add name="RequestStreamDumpModule" type="Memba.FileUpload.RequestStreamDumpModule, Memba.FileUpload, Version=1.0.0.0, Culture=neutral, PublicKeyToken=a4ae091aa8097a5a"/>

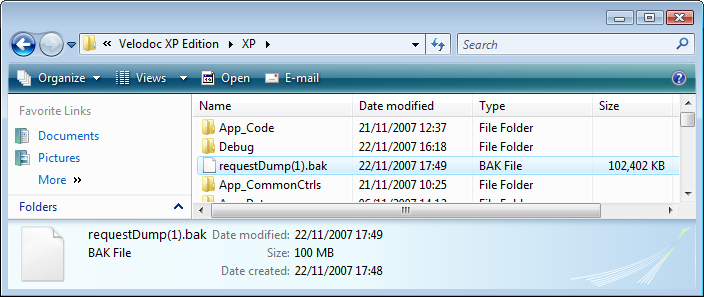
This will generate a dump of the request in the web application directory unless you also configure the key dumpFileLocation in the <configuration><appSettings> section of web.config.

<add key="dumpFileLocation" value="14" />

Launch the ~/Debug/Download.aspx page of web site project in the Velodoc XP Edition solution and upload a large file.

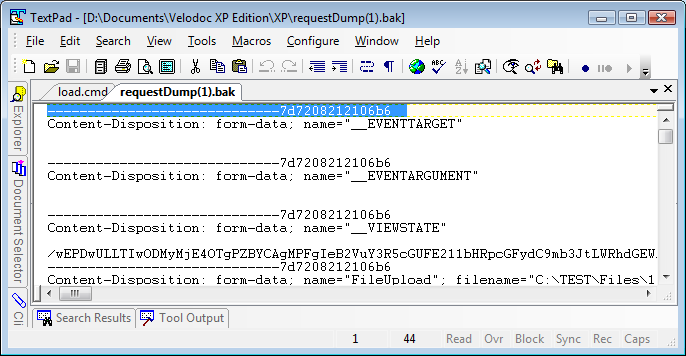


At the end of the upload, a dump of the request is created either in the web application directory or in the location specified by dumpFileLocation:



IMPORTANT: Now that you have a dump request, restore the Upload Http Module.

Use a text editor to open the dumped request file and copy the multipart boundary:



Create a command file called “load.cmd” and edit it with a text editor. You need the following command line – on one single line – followed by a pause.

"C:\TEST\TinyGet\TinyGet.exe"

-status:200

-verb:POST

-srv:localhost

-port:49855

-uri:/XP/Debug/Download.aspx?muid=e1291edcd06340dcb640fe51bfc7ec41

-frb:"D:\Documents\Velodoc XP Edition\XP\requestDump(1).bak"

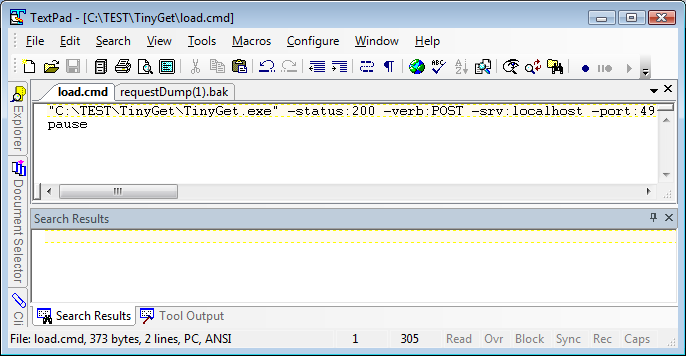
-reqheaders:"Content-Type: multipart/form-data; boundary=---------------------------7d7208212106b6a\n"

-threads:10

-loop:10

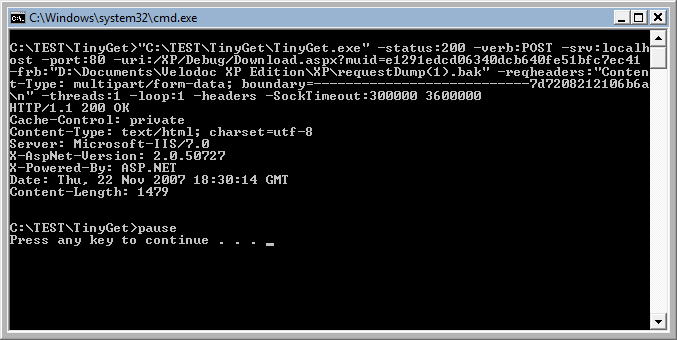
-headers

-SockTimeout:300000 3600000

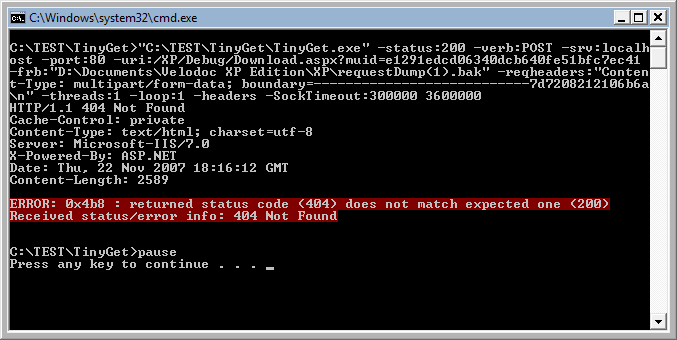


* **status** refers to the http status which you expect in return;
* **verb** should be POST;
* **srv** is the ip address of the server you target;
* **port** is the port number on the server you target;
* **uri** is the path to the upload page, which in our example happens to be named Download.aspx; Do not forget the muid;
* **frb** is the local path to the request dump file;
* **reqheaders** is where you paste the multipart boundary copied from the request dump file. Be careful: you need to remove two dashes. In our example, the boundary in the request dump file has 29 dashes. In the command line, it only has 27 dashes. Do not forget to add a line break at the end (\n);
* **threads** is the number of simultaneous threads executing uploads;
* **loop** is the number of times each thread executes an upload;
* **headers** just displays the headers in the console;
* **socktimeout** gives time for the uploads to execute.

Run load.cmd. When successful, you get the following result:



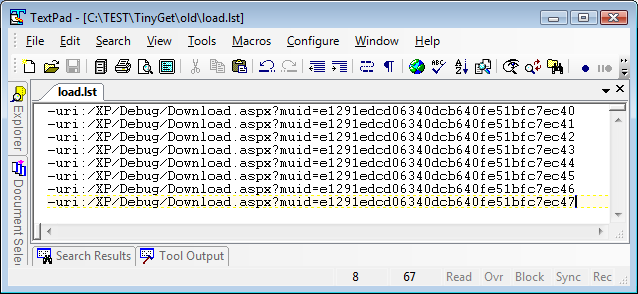
A bad request looks like follows:



Note that you will have to do this each time you test on a new web server because ASP.NET ViewState in the request is encrypted using a machine key. You can also configure the machine key as explained in <http://msdn2.microsoft.com/en-us/library/ms998288.aspx>. You can find an online generator at <http://www.orcsweb.com/articles/aspnetmachinekey.aspx>. Then you should be able to use your dump file with any web server implementing this machine key.

The fact that you use the same muid for all requests should not be a problem here but if you do not feel comfortable with this you can use a script list in load.cmd:

1. replace:
   * -uri:/XP/Debug/Download.aspx?muid=e1291edcd06340dcb640fe51bfc7ec41  
     with
   * -script:"C:\TEST\TinyGet\load.lst"
2. Only use a single thread
3. Create load.lst with the following content:



Run load.cmd.

# New in version 1.1

We have essentially added unit tests for the WCF streaming service.