

Modelit
Elisabethdreef 5
4101 KN Culemborg
The Netherlands
+31(345)531717



info@modelit.nl
www.modelit.nl

User Guide for the Modelit Matlab Webservice Toolbox

Authors N.J. van der Zijpp
K.J. Hoogland

Latest revision: 20210917

Revision history

- 20210917 Evaluation version is precompiled with Matlab 2021a
`editClasspath.m` updated.
- 20160520 Default startup.m included with the toolbox.
- 20160308 New load balance feature: forward request to port on different server using IP address.
- 20150825 Update of user guide. Chapters "Getting started" and "Preparing the Matlab development environment" added.
- 20150825 Inclusion of new utilities:
- `json2struct.m` quick and easy conversion of JSON strings to Matlab array of name/value pairs. Values may be string, numeric, true, false, null, JSON expressions or arrays of values.
 - `urlread_async.m` utility for the asynchronous consumption of webservices. This makes it possible to call multiple webservices in parallel.
 - `matlabprogress.m` progress bar for `urlread_async` featuring matlab waitbar.
 - `modelitprogress.m` progress bar for `url_async` featuring Modelit nested waitbar.
 - `serialize_v2.m` improved version of `serialize.m`. Faster, does not write to disk, and leads to smaller data transfers
 - `deserialize_v2.m` counterpart of `serialize_v2`.
- 20150228 Add note on new Matlab feature "javaclasspath.txt". Full source code now available for examples. Instructions for deployment in compiled mode added.
- 20140212 Textual updates.
- 20131002 Response and request parameters can now be set. Examples added for xml, html and pdf output. Toolbox has been tested with Tomcat versions 7.0 and 8.0.
- 20120809 Refreshed screenshots, use consistent url's in example.
- 20120125 Minor revisions.
- 20110920 Update to Matlab programming reference
- 20110902 Added: info on using Matlab `urlread`.
Added: configure firewall on Win 7.
Added: configuring Dual Wan router.
- 20110531 Minor revision, refresh screen dumps.
- 20100918 Corrections.
Added: verification using proxy server.

- 20100818 Minor revision (chapter 8).
- 20100723 Added example for 2 webservices on 1 computer.
- 20091014 Added requirements and notes on troubleshooting.
- 20090801 Initial version of the toolbox.

Contents

1	Introduction	1
1.1	About the Modelit Matlab Webservice Toolbox	1
1.2	Deployment options	2
1.3	Setting up of a web-based solution	3
1.4	Requirements.....	5
1.5	Requirements for development environment	5
1.6	Requirements for deployment environment.....	5
1.7	Difference between trial version and full version of the toolbox .	5
2	Getting started	6
2.1	Contents of the install package.....	6
2.2	Quick install instructions.....	6
2.3	Step by step install instructions	6
3	Installing Apache Tomcat on the Webserver Machine	7
3.1	Download Apache Tomcat.....	7
3.2	Run the Apache Tomcat Installer	8
3.3	Verifying the Tomcat Installation	9
3.4	Changing Tomcat after installation.....	10
3.4.1	Adjusting the Tomcat port by editing the server.xml .	10
3.4.2	Add/change users in the tomcat-users.xml	10
4	Adding the Modelit Matlab Webservice Component to Apache Tomcat	12
4.1	Copying the Web-inf directory to your disk	12
4.1.1	Alternative location of WEB_INF	12
4.2	Configuring the servlet.....	13
4.3	Setting up multiple webservices on one computer.....	17
4.4	Load balancing	17
4.5	Failover	19
4.6	Verifying the servlet installation	19
4.7	Troubleshooting	19
5	Preparing the Matlab development environment.....	20
5.1	Method 1: use the default startup.m that comes with the toolbox	20
5.2	Method 2: integrate the toolbox with your existing project.....	20
5.2.1	Installing m-files.....	20
5.2.2	Setting the static Java classpath for Matlab sessions .	20
5.3	Verifying the Webservice installation	23
5.4	Troubleshooting	23
6	Matlab examples	24
6.1	HTML example	25
6.2	XML example	28
6.3	HTML example with graph.....	29
6.4	PDF example.....	31
6.5	Communicate between Matlab sessions	32
7	Using the Webservice Toolbox with compiled Matlab code	36
7.1	Reason for deployment in compiled mode.....	36

7.2	First note: Setting java classpath in the deployment environment	36
7.3	Second note: prevent the webservice from exiting	37
7.4	Code example	38
8	Programming reference.....	40
8.1	createMatlabServer.....	40
8.2	ServerEvent Object.....	41
8.3	Troubleshooting	44
8.3.1	Java exception	44
8.4	Callback cannot be reached for specific ports	44
9	Windows Firewall Configuration on the Webservice Machine.....	45
9.1	Configuration on Windows XP.....	45
9.1.1	Opening a port	45
9.1.2	Restricting access	46
9.1.3	Adding a range of IP-addresses (subnet mask)	46
9.1.4	Active settings.....	46
9.2	Configuration on Windows 7.....	47
9.2.1	Interactive setup	47
9.2.2	Command line setup	47
10	Setting up the TCP/IP configuration on the server	48
10.1	TCP/IP properties	48
10.2	Verifying TCP/IP properties	49
11	Setting up port forwarding on the network router.....	51
11.1	Introduction.....	51
11.2	Step 1: disable DHCP on the server	51
11.3	Example: setting up port forwarding on Draytek Vigor 2920 ...	51
11.4	Advanced topic: Setting up redundant servers and WAN's	52
12	Verifying the Matlab Webservice installation	54
12.1	Verification with a web browser	54
12.2	Verification with Firefox Poster	54
12.3	Verification from Matlab	55
12.4	Verification using proxy server	55
13	Example: Integrating Matlab in web pages using XMLHttpRequest....	56

1 Introduction

1.1 About the Modelit Matlab Webservice Toolbox

The Modelit Matlab Webservice Toolbox makes it possible to create webservices based on Matlab code in an easy manner at the lowest possible cost.

In addition to this the toolbox provides utilities that will help you implement these webservices, like utilities for creating XML output from Matlab data, encapsulating Matlab figures in HTML, converting JSON strings to Matlab, serialize and deserialize to and from base64 encoded ASCII, or parallelize tasks by making asynchronous calls to webservices. The toolbox provides a number of examples that will help you getting started.

This user guide provides a step-by-step instruction for installing the toolbox and should allow anyone with basic Matlab knowledge to create and deploy Matlab based webservices. In addition the user guide provides many useful tips and tricks that we have learned over the years.

To get started right away, skip to chapter 2.

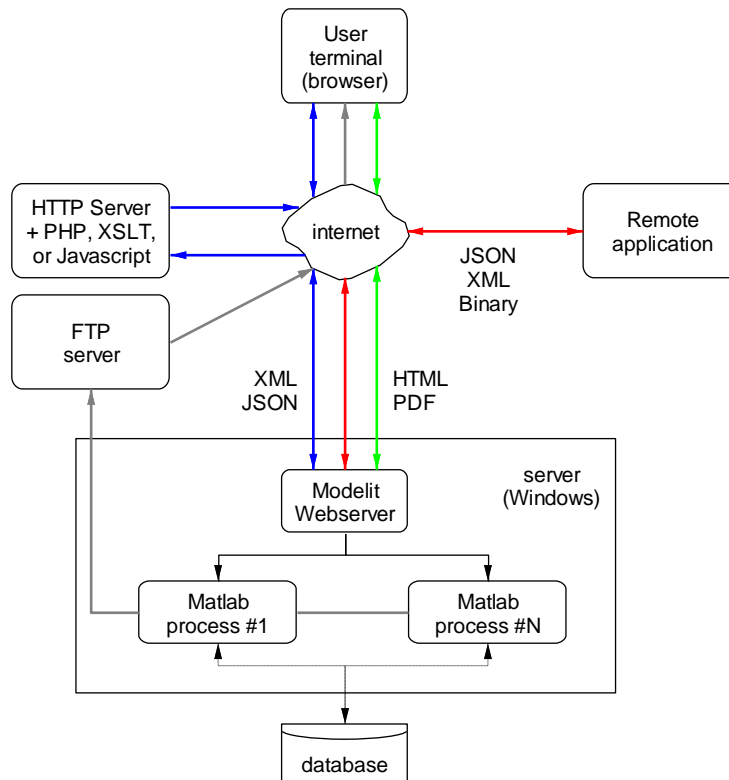


Figure 1: Deployment options for the Matlab Webservice Toolbox

1.2 Deployment options

Converting your Matlab code to a webservice makes it possible that this code is used anywhere on the world. Your software can be called from a HTTP server, directly from a browser, or by another computer program. Apart from this interoperability advantage, making software available as a webservice may help you comply with corporate IT requirements, allows you to protect sensitive data, or simply increase computing capacity by running many instances of single webservice at the same time.

The toolbox has many deployment options (see Figure 1), and new ones can be added upon request. The table below shows options that have been applied in the past.

Deployment option	Typical setup
Browser to server, direct	<ul style="list-style-type: none"> • Browser makes HTTP-GET request to Matlab server; • Matlab callback generates HTML code that is presented to the user. Utilities to encapsulate Matlab figures in HTML pages are included in the toolbox. <p>Alternative use:</p> <ul style="list-style-type: none"> • Matlab generates dynamic PDF documents that are opened in the browser of the user.
Application to server	<ul style="list-style-type: none"> • Application makes HTTP-GET or HTTP-POST request to Matlab server; • In case of HTTP-POST request: common data formats are XML, JSON or binary; • Matlab server replies with XML, JSON or binary (serialized Matlab) message; <p>Special case: Webservice is called from Matlab using <code>urlread</code>:</p> <ul style="list-style-type: none"> • The toolbox includes utilities to serialize and deserialize Matlab variables. This makes it possible to exchange Matlab variables between Matlab server and Matlab session. • The toolbox includes a utility "<code>urlread_async</code>". This makes it possible to parallelize tasks.
Browser to server, via external HTTP server	<ul style="list-style-type: none"> • Browser makes request to HTTP server; • HTTP server runs PHP, JavaScript or XSLT and makes HTTP-GET or HTTP-POST request to Matlab server; • Matlab server replies with JSON or XML message; • Example: www.tripcast.nl

Things the Modelit Webservice Toolbox cannot do

At this point the Webservice Toolbox cannot port complete Matlab applications to web based applications, nor does it contain a web based equivalent of Matlab Guide. The toolbox can be used to create webservices that implement GUI callbacks, but the GUI front end must be implemented in HTML, JavaScript, PHP or the like.

1.3 Setting up of a web-based solution

Figure 2 shows a typical setup of a web based solution and shows the components that are involved and how they interact. The figure shows that a web based solution uses various components, like as HTTP-Server, a Router, a Local (software) Firewall, Apache Tomcat, and a Matlab component. All these components require attention. The Modelit Webservice Toolbox has been developed to the current version over a number of years. It consists of a collection of software components , but even more so it represents the know-how that is required to set up stable and performing web based solutions using Matlab. This user guide aims to summarize this known-how, and is not limited to Matlab issues only.

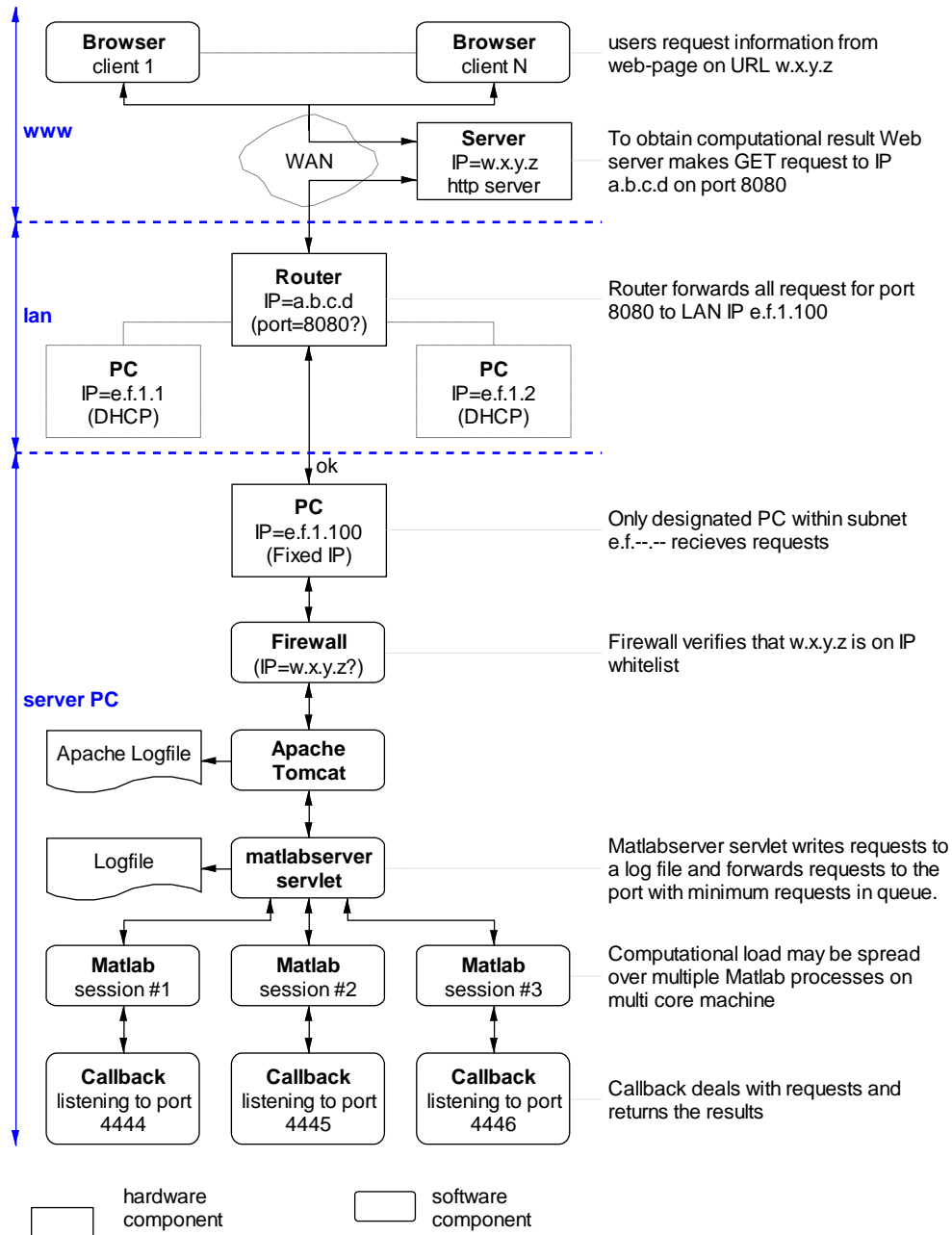


Figure 2: Typical setup of a system using the Modelit Matlab Webservice toolbox

1.4 Requirements

Typically, webservices are deployed on a dedicated server or on a server in the cloud as provided by (for example) Amazon EC2, while the software is developed elsewhere. For this reason requirements are listed separately for the development environment and the deployment environment.

Obviously, software can be developed and deployed (or tested) on a single computer, provided this computer meets requirements for development and deployment. In fact, this user guide assumes you will be doing this.

1.5 Requirements for development environment

To develop a webservice software package ready for deployment the following is needed:

- A Windows or Linux computer with Matlab 2006b or higher installed on it;
- (optional) Matlab compiler (*);
- Modelit Matlab Webservice toolbox.

(*) Matlab compiler is not a requirement, but is recommended. Otherwise each webservice instance will occupy a Matlab license. In compiled mode, multiple webservice instances can run at no additional costs.

1.6 Requirements for deployment environment

To deploy this webservice software package you need the following:

- A Windows or Linux machine with Matlab or Matlab Runtime installed (*);
- A connection to internet, preferably using a fixed IP-address;
- Access to any routers that connect this computer to the internet so that you can setup port forwarding;
- Apache Tomcat 6.0 software or higher (version 8 recommended, available for free at <http://tomcat.apache.org>);

(*) Matlab Runtime is obtainable for free at:

<http://nl.mathworks.com/products/compiler/mcr/>.

Select the version that Matches the Matlab compiler that was used.

1.7 Difference between trial version and full version of the toolbox

The trial version is fully functional but contains pcode with a limited usage time only. The pcode is generated with Matlab 2013a, but should be compatible with Matlab versions 2006a and above.

2 Getting started

2.1 Contents of the install package

The install package is a zipfile named matlabserver.zip (full version) or matlabserver_pcode.zip (trial version).

2.2 Quick install instructions

This zipfile has two subdirectories:

- matlabserver. This directory including its subdirectories is added to the Apache Tomcat installation. See chapter 4.
- matlabcode. This directory including and its subdirectories are added to the Matlab path. See chapter 5.

2.3 Step by step install instructions

The next chapters explain the following tasks with step by step instructions and verifications:

- Installing Apache Tomcat (chapter 3);
- Adding the Modelit Matlab Webservice Component to Apache Tomcat (chapter 4)
- Preparing the Matlab development environment (chapter 5)
- Creating an example Matlab program;
- Configuring the Windows Firewall;
- Setting up port forwarding for a local network.

After completing these tasks the example Matlab program that you have created will be ready to be called from any internet browser.

3 Installing Apache Tomcat on the Webserver Machine

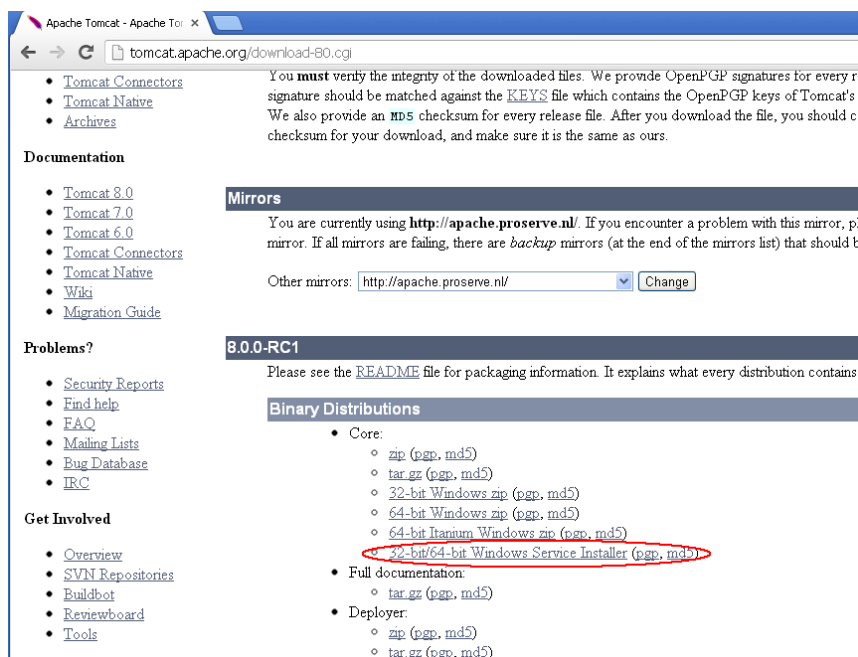
3.1 Download Apache Tomcat

Go to <http://tomcat.apache.org>



Navigate to the Tomcat 8.0 download section by selecting Tomcat 8.0 from the download menu on the left side of the webpage. Now, under the 8.0 section download the Windows Service Installer, save it to disk.

Note: the Webservice toolbox requires Tomcat 6.0 or higher

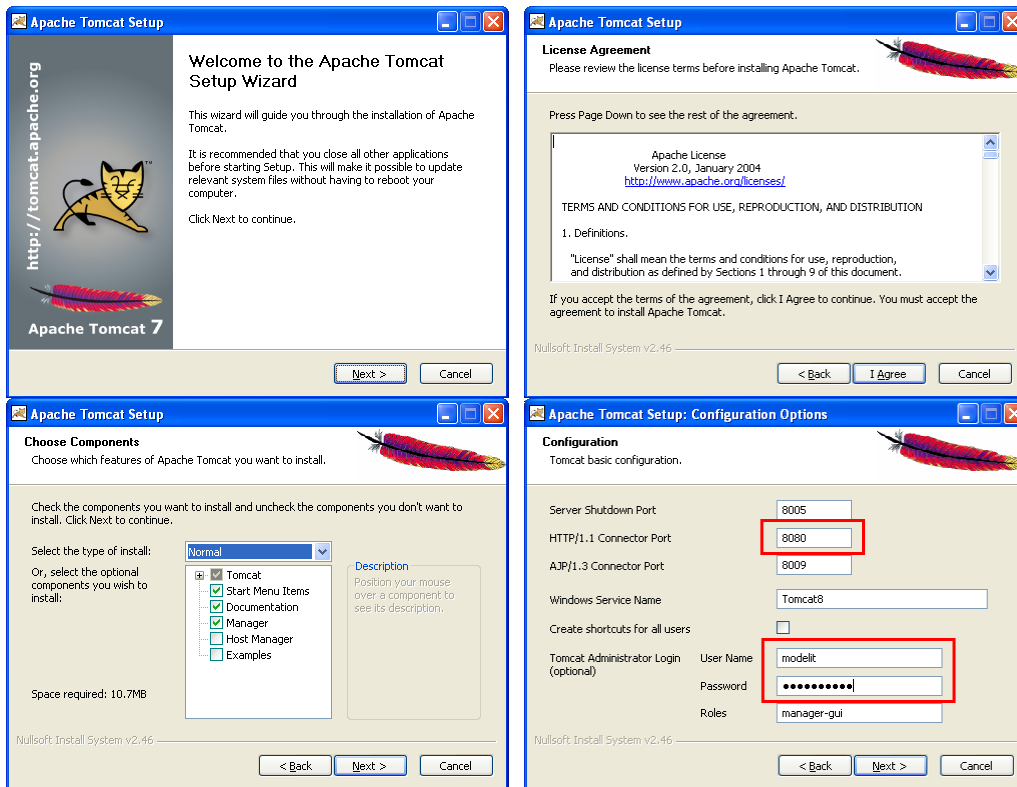


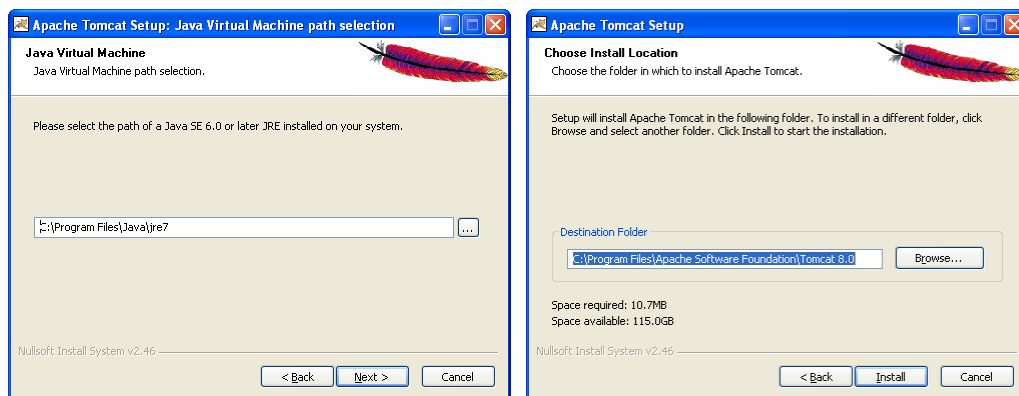
3.2 Run the Apache Tomcat Installer

Run the installer. A number of screens will appear. In general, only the screen "Configuration Options" requires input.

The user is prompted for a port number, username and password and the path to the Java Runtime Environment (JRE).

- The port number determines the port number that should be used to access your Matlab callback function. To change it afterwards, rerun the installer or edit the server.xml file in the Tomcat/conf directory, see below.
- The username and password are optional. Specify these if you want to inspect the server status and manage the installed applications from a web-console, see "Add/change users in the tomcat-users.xml" below. Use a User Name of choice.
- In general the JRE will already be installed. If not go to www.java.com and download and install the required JRE.








3.3 Verifying the Tomcat Installation



Monitor Tomcat

- Select  ;
- Select "All programs/Apache Tomcat 8.0/Monitor Tomcat";
-  The Tomcat icon becomes visible;
- Initially Tomcat will be in idle mode. Double click the icon in the tray. The Tomcat console becomes visible;
-  Press the start button;
- Tomcat is now running;
- The Tomcat console can be closed.

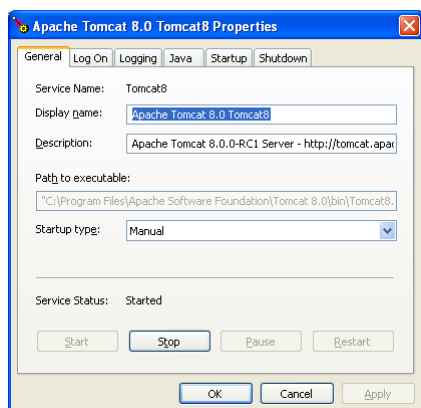
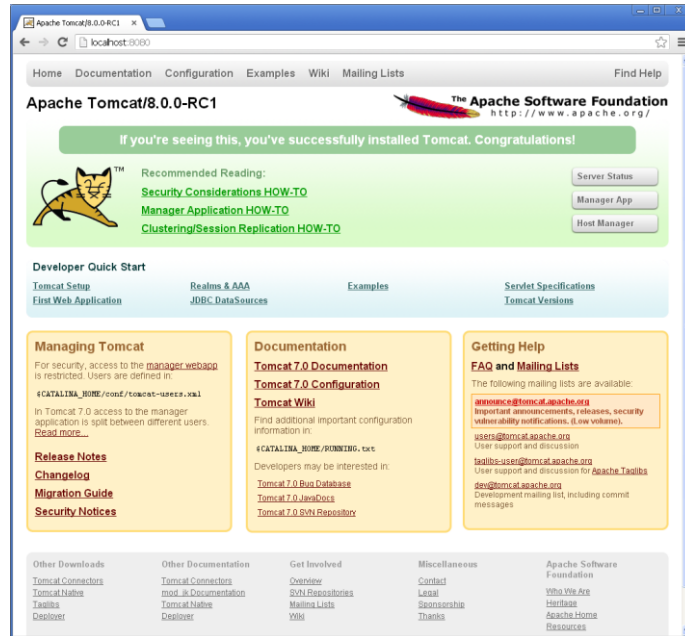


Figure 3: Tomcat Console.

Now verify the installation by typing the following address in the browser's address bar: <http://localhost:8080> for the standard installation.



If this screen appears, you have successfully installed Apache Tomcat, and you are ready to install the Modelit Matlab Webservice Toolbox as described in chapter 4.

3.4 Changing Tomcat after installation

3.4.1 Adjusting the Tomcat port by editing the server.xml

By default Apache Tomcat is configured to run on port 8080. After installation the port can be changed by editing the server.xml file which is located in \$TOMCAT_HOME/conf (e.g. C:\Program Files\Apache Software Foundation\Tomcat 8.0\conf). To change the port change the value 8080 in the following block of code into the desired port number.

listing of ..\Tomcat 8.0\conf\server.xml

```
<!-- A "Connector" represents an endpoint by which requests are received
and responses are returned. Documentation at :
Java HTTP Connector: /docs/config/http.html (blocking & non-blocking)
Java AJP Connector: /docs/config/ajp.html
APR (HTTP/AJP) Connector: /docs/apr.html
Define a non-SSL HTTP/1.1 Connector on port 8080
-->
<Connector port="8080" protocol="HTTP/1.1"
connectionTimeout="20000"
redirectPort="8443" />
<!-- A "Connector" using the shared thread pool-->
```

-end of listing-

3.4.2 Add/change users in the tomcat-users.xml

To be able to inspect the server status and manage apps from a web-based console (accessible at <http://localhost:8080>) a user with a password and role 'manager-gui' must be defined in the tomcat-users.xml

in the Tomcat conf directory. If you did not specify the Tomcat Administrator Login during installation you can add this information manually.

listing of ..\Tomcat 8.0\conf\tomcat-users.xml

```
<tomcat-users>
<user name="modelit" password="mypassword" roles="manager-gui" />
<!--
NOTE: By default, no user is included in the "manager-gui" role required
to operate the "/manager/html" web application.  If you wish to use this app,
you must define such a user - the username and password are arbitrary.
-->
<!--
NOTE: The sample user and role entries below are wrapped in a comment
and thus are ignored when reading this file. Do not forget to remove
<!-- ... --> that surrounds them.
-->
<!--
<role rolename="tomcat"/>
<role rolename="role1"/>
<user username="tomcat" password="tomcat" roles="tomcat"/>
<user username="both" password="tomcat" roles="tomcat,role1"/>
<user username="role1" password="tomcat" roles="role1"/>
-->
</tomcat-users>
```

-end of listing-

After restarting Tomcat the Server status and App Manager are accessible to the defined user(s).

4 Adding the Modelit Matlab Webservice Component to Apache Tomcat

4.1 Copying the Web-inf directory to your disk

Unzip the file MatlabServer.zip to \$TOMCAT_HOME\webapps\

After this, the directory structure should look like this:

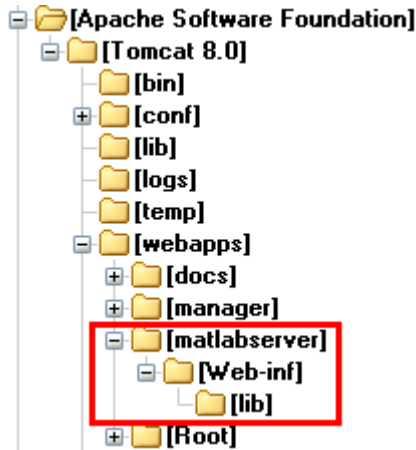


Figure 4: Directory structure after copying matlabserver directory.

This installs a webservice that can be accessed from any browser as:
 127.0.0.1:8080/matlabserver/myExample (*)

(*) assuming you installed Tomcat with port 8080.

See also: "verifying the servlet installation" (section 4.6).

The remainder of this chapter shows how to change the parameters of the servlet. If you are happy with the default settings you may skip this to a later date.

4.1.1 Alternative location of WEB_INF

In this guide we will assume that you have copied Web_inf to:
 \$TOMCAT_HOME\webapps\matlabserver\Web-inf.

As a result the path to any Matlab webservice defined on this computer will look like:

"<IPADDRESS: PORT>/matlabserver/<name>".

With <name> a service name that is defined in web.xml.

You may however place the "Web-inf" folder anywhere in Tomcats "webapps" folder. For example, if you copy web_inf to:

\$TOMCAT_HOME\webapps\algorithm\Web-inf

Your webservice is accessible as:

"<IPADDRESS: PORT>/algorithm/<name>".

As a special case you may copy "web_inf" to:

\$TOMCAT_HOME\webapps\ROOT\Web-inf

In this case your webservice is accessible as:

"<IPADDRESS: PORT> /<name>".

4.2 Configuring the servlet

The step above will install a software component known as a *servlet*. This servlet receives http requests on the connector port (for example 8080) and forwards these to Matlab callback functions on one or more ports of choice.

The file

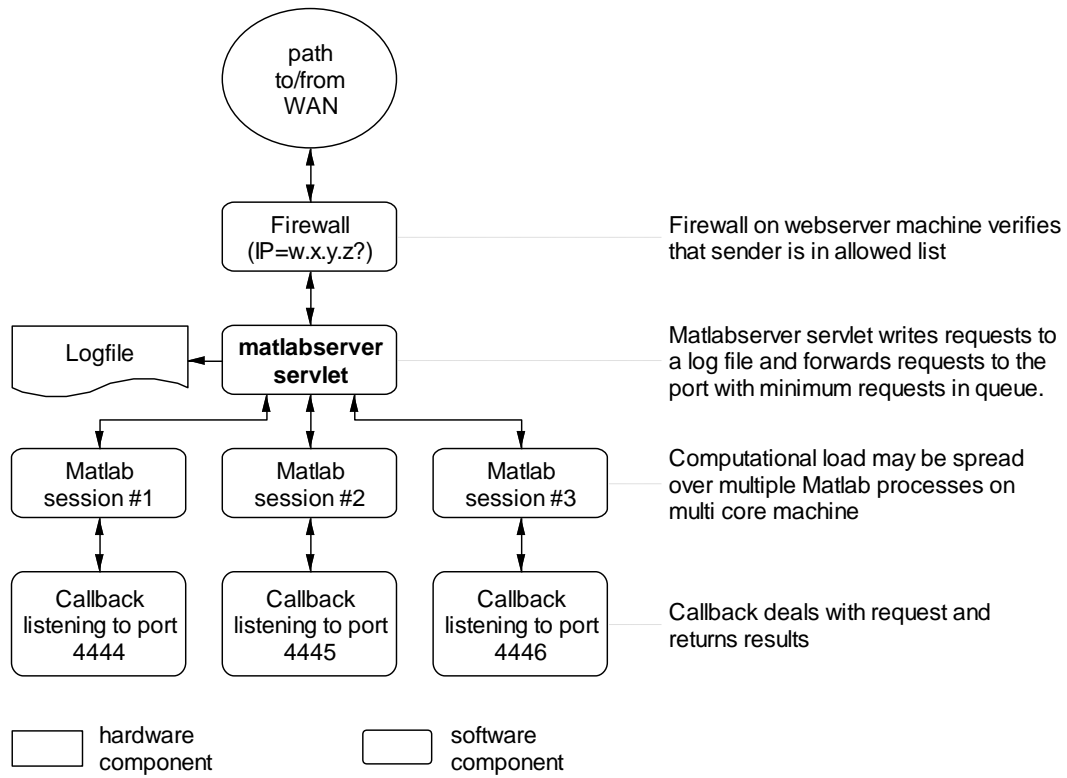
\$TOMCAT_HOME\webapps\matlabserver\WEB-INF\lib\MatlabServer.jar
contains the java classes used by matlabserver.

The file

\$TOMCAT_HOME\webapps\matlabserver\WEB-INF\web.xml
contains the configuration data for the matlabserver servlet that you might want to modify.

You can specify the port numbers the servlet forwards its requests to. If multiple ports are specified, the incoming requests will be distributed over these ports.

This provides a load balancing mechanism: The computational load is distributed over multiple processes. Typically each Matlab session will listen to one port. Ports can be defined in the red area in the listing.



listing of ..\Tomcat 8.0\webapps\matlabserver\Web-inf\web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<web-app xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  version="2.5">

  <display-name>Modelit Matlab server</display-name>
  <description>
    Connecting java servlets with Matlab
  </description>

  <servlet>

    <servlet-name>servlet1</servlet-name>
    <servlet-class>nl.modelit.matlabserver.MatlabServlet</servlet-class>

    <init-param>
      <param-name>port</param-name>
      <param-value>4444,4445,192.168.3.100:4444</param-value>
    </init-param>

    <init-param>
      <param-name>mode</param-name>
      <param-value>normal</param-value>
    </init-param>

    <init-param>
      <param-name>log</param-name>
      <param-value>2000</param-value>
    </init-param>

    <init-param>
      <param-name>queue</param-name>
      <param-value>4</param-value>
    </init-param>

  </servlet>

  <servlet-mapping>
    <servlet-name>servlet1</servlet-name>
    <url-pattern>/myExample</url-pattern>
  </servlet-mapping>

</web-app>
```

-end of listing-

Configurable parameters	Description
 servlet-name	Make sure servlet/servlet-name corresponds to: servlet-mapping/servlet-name.
 url-pattern	<p>Specify the path where the url where the webservice will be available.</p> <p><i>Notes on urlpattern:</i> To access the webservice for this example the url that must be entered is: http://localhost:8080/matlabserver/myExample</p> <p>This is because the "Web-inf" directory resides in "Tomcat 6.0\webapps\matlabserver".</p> <p>Change the directory name "matlabserver" to another to change the url. For example, if the directory name "matlabserver is" replaced by "services" the url becomes: http://localhost:8080/services/myExample</p> <p>Move the contents of "Web-inf" to "Tomcat 6.0\webapps\Root\Web-inf" and the url will change to: http://localhost:8080/ myExample</p>
 port	Comma separated list with port numbers. Specify at least one port number. External calls will be distributed over all specified ports. N.B. If no callback is specified for a specific port a timeout will occur on the first call. Subsequent calls will no longer be directed to this port. Therefore there is not too much harm in specifying more ports than you initially need.
 IP address	By default, requests will be forwarded to port number specified at "port" at localhost. To forward requests to another physical machine, include the IP address of this machine by prepending the IP address of the alternate machine to "port" (the syntax is IP-address:port).
 mode	"normal" or "debug". If debug is specified additional information is written to the tomcat8-stdout.log file in the Tomcat logs directory.
 queue	Maximum queue length for every port number.
 log	Maximum length of logging message. If the length of a request exceeds this number, all characters beyond position "log" will not be included in the log file.

4.3 Setting up multiple webservices on one computer

It is possible to specify different webservices that run on a single machine. Adapt the file web.xml in the following way:

listing of ..\Tomcat 8.0\webapps\matlabserver\Web-inf\web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<web-app xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  version="2.5">

  <display-name>Modelit matlab server</display-name>
  <description>
    Connecting java servlets with Matlab
  </description>

  <servlet>
    <init-param>
      <param-name>port</param-name>
      <param-value>4444,4445</param-value>
    </init-param>

    <servlet-name>servlet1</servlet-name>
    <servlet-class>nl.modelit.matlabserver.MatlabServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>servlet1</servlet-name>
    <url-pattern>/myExample</url-pattern>
  </servlet-mapping>

  <servlet>
    <init-param>
      <param-name>port</param-name>
      <param-value>4446,4447</param-value>
    </init-param>

    <servlet-name>servlet2</servlet-name>
    <servlet-class>nl.modelit.matlabserver.MatlabServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>servlet2</servlet-name>
    <url-pattern>/yourExample</url-pattern>
  </servlet-mapping>

</web-app>
```

-end of listing-

4.4 Load balancing

Load balancing is the process of distributing tasks over different computers or (as in the current case) processes. If multiple ports are specified in the web.xml (see above) the incoming http requests will be distributed over these ports. The Webservice toolbox provides a mechanism to implement Matlab callbacks that listen to a specific port (see chapter 6). Multiple Matlab processes can be started, where each

process contains a callback that listens to one of the ports specified in web.xml. In this way http requests are processed in parallel by different Matlab processes. Especially on multi-core processors this increases capacity and reduces response times.

Queuing

Under heavy loads the webservice toolbox will behave as follows:

- The server will first try to find a "free" port to handle a specific request, where "free" means that the process is not currently waiting for a reply on a message that was sent to this port;
- If all ports are "busy" it will add the request to the end of the queue with the minimum number of request in it;
- If at the time of a request all queue lengths exceed a specific number (currently fixed at 4) for all ports the server will immediately return the request with the message "Server Busy".

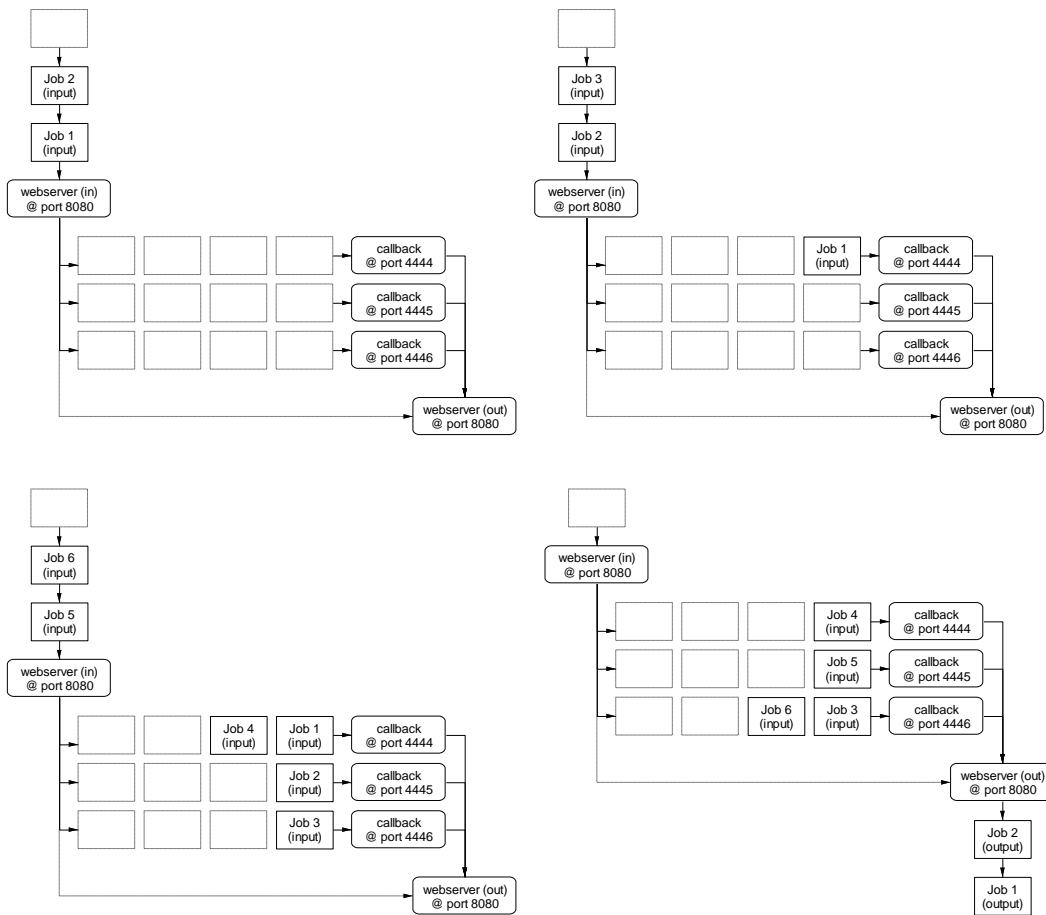


Figure 5: *Demonstration how a platoon of jobs propagates through a setup of 3 parallel "pipelines". Each queue contains a maximum of 4 jobs. If all queues are full, the webservice toolbox will bypass the queue (indicated by the dotted arrow) and immediately return a "no capacity" message.*

4.5 Failover

When there is no reply to a request that is forwarded to a port, it is likely that no callback is listening to this port. In this case, also future requests will remain unanswered. For this reason this port is removed from the list of ports that are considered.

Failover recovery

Removal of a non-responding port is permanent. The only way to restore all ports is to restart Apache Tomcat.

4.6 Verifying the servlet installation

To verify the servlet installation call the servlet with for example two arguments:

<http://localhost:8080/matlabserver/myExample?arg1=1&arg2=2>

(again replace 8080 with the port number on which Tomcat is running).

The message "no port available" appears because no Matlab session is listening to any of the in the web.xml defined ports.



4.7 Troubleshooting

If you receive a 404 error like in the figure below, the most likely cause is that the url you are viewing is not consistent with the settings in web.xml (see section 4.2). In particular note that the url must be typed case sensitive!



5 Preparing the Matlab development environment

5.1 Method 1: use the default startup.m that comes with the toolbox

For a quick try of the toolbox, follow these steps:

- Make sure that the web-inf directory is installed (see chapter 4);
- Locate the directory "matlabcode" in the Modelit Matlab Webservice Toolbox install package "matlabserver.zip" (full version) or "matlabserver_pcode.zip" (trial version);
- Copy the contents of this directory to a location of your choice;
- Create a desktop icon that starts Matlab in this location;
- Start Matlab from the desktop shortcut and follow instructions.

You may now proceed to "Verifying the Webservice installation" (section 5.3)

5.2 Method 2: integrate the toolbox with your existing project

5.2.1 *Installing m-files*

The Modelit Matlab Webservice Toolbox install package is a zip file MatlabServer.zip (or matlabserver_pcode.zip for trial version).

The source code is organized in three directories:

- matlabcode. this directory contains a number of m files that represent the core functionality of the toolbox;
- matlabcode/examples. this directory contain example programs as used in this guide;
- matlabcode/utils. This directory contains utilities that may be useful when implementing you own webservices.

These separate directories need to be added to your Matlab path. The Matlab command for this is "addpath". It is recommended that you edit your Matlab startup.m file so that it includes the commands that add these directories to your path. The toolbox comes with a default startup.m that already sets the correct path.

5.2.2 *Setting the static Java classpath for Matlab sessions*

Matlab loads its static Java classpath at startup from the file "classpath.txt". This file is usually located in the directory that is returned by the command "matlabroot". Normally Matlab users do not need to change the static classpath, but users of the Modelit Webservice Toolbox need to add the full path to MatlabServer.jar to this path.

For this purpose the file classpath.txt must be copied from the Matlabroot directory to the Matlab startup directory before any edits are made, otherwise these edits will affect all users that share the matlabroot.

Note for Matlab versions 2012b and up:
As of Matlab version 2012b Matlab ignores any copies of "classpath.txt" in the Matlab startup directory. Instead Matlab looks for a file called "javaclasspath.txt". So the file classpath.txt in Matlabroot must be copied to the Matlab startup directory and renamed to javaclasspath.txt.

This section describes two ways to apply these edits. The first way is to run a script called "editClasspath.m". The second way is to manually edit the classpath file.

Running editClasspath.m

The function "editClasspath.m" reads the file classpath.txt from Matlabroot and appends MatlabServer.jar tot the end of the classpath and saves a modified copy of classpath.txt (Matlab 2012b and up: javaclasspath.txt) in the startup directory. For most users this is an easy and adequate way to edit the static Java classpath.

Proceed as follows:

- Start a Matlab session using the startup directory that you intend to use for the project that involves the Modelit Matlab Webservice Toolbox;
- Type `addpath(<dirName>)` on the command prompt, where `<dirName>` is the directory where the files `MatlabServer.jar` and `editClasspath.m` reside (for example: `c:\Program Files\Apache Software Foundation\Tomcat 8.0\webapps\ROOT\WEB-INF\lib\`);
- Run `editClasspath` from the command prompt;
- Close your current Matlab session and restart Matlab;
- Type "javaclasspath" on the command prompt and check that `MatlabServer.jar` including its path is listed under the Static Java Path.

Matlab versions prior to 2012b: editing the classpath.txt file manually

If for some reason the above does not work for you, for example, you have already made modifications to the classpath.txt file that should not be made undone, you may manually edit the classpath.txt file.

Proceed as follows:

- Start a Matlab session using the startup directory that you intend to use for the project that involves the Modelit Matlab Webservice Toolbox;
- Locate the classpath.txt file by typing `"which('classpath.txt')"` on the command prompt;
- If this file is located in Matlabroot, make a copy of this file in the Matlab startup directory, otherwise your edits will affect all Matlab users on your system that share Matlabroot;
- Edit the file classpath.txt by typing `"edit classpath.txt"`;

- Append the full path to the MatlabServer.jar to the file, this file resides in \$TOMCAT_HOME\webapps\matlabserver\WEB-INF\lib\
For example in the default installation add the following line:
C:\Program Files\Apache Software Foundation\Tomcat
8.0\webapps\matlabserver\WEB-INF\lib\MatlabServer.jar
- Save classpath.txt in the *current directory*, i.e. the directory that appears after typing pwd on the command line;
- Restart the Matlab session to activate new java classpath;
- Type "javaclasspath" on the command prompt and check that MatlabServer.jar including its path is listed under the Static Java Path.

Matlab versions 2012b and up: editing the javaclasspath.txt file manually

If for some reason the above does not work for you, for example, you have already made modifications to the classpath.txt file that should not be made undone, you may manually edit the classpath.txt file.

Proceed as follows:

- Start a Matlab session using the startup directory that you intend to use for the project that involves the Modelit Matlab Webservice Toolbox;
- Locate the javaclasspath.txt file by typing "which('javaclasspath.txt')"
on the command prompt;
- If this file is not found, rename the file "classpath.txt" in Matlabroot to "javaclasspath.txt" in de startup directory.
- Edit the file javaclasspath.txt by typing "edit javaclasspath.txt";
- Append the full path to the MatlabServer.jar to the file, this file resides in \$TOMCAT_HOME\webapps\matlabserver\WEB-INF\lib\
For example in the default installation add the following line:
C:\Program Files\Apache Software Foundation\Tomcat
8.0\webapps\matlabserver\WEB-INF\lib\MatlabServer.jar
- Save javaclasspath.txt in the *current directory*, i.e. the directory that appears after typing pwd on the command line;
- Restart the Matlab session to activate new java classpath;
- Type "javaclasspath" on the command prompt and check that MatlabServer.jar including its path is listed under the Static Java Path.

5.3 Verifying the Webservice installation

Open a Matlab session.

From the Matlab command line, type:

```
server = createMatlabServer(4444)
```

You should get something similar to the following response:

```
server =  
nl.modelit.matlabserver.MatlabServer@599d5b
```

If you get this response, you are ready to create your first Matlab-webservice. Proceed to chapter 6.

5.4 Troubleshooting

- Verify that you have saved the modified file classpath.txt in the correct location by typing: `which('classpath.txt')` on the Matlab command line. Remember that all java classes are cases sensitive;
- Verify that you have restarted Matlab after changing classpath.txt. Changes in classpath.txt take effect after restarting Matlab;
- Verify that you are using Matlab 2008 or later. The Webservice toolbox has not been tested with earlier Matlab versions;
- If you obtain the HTTP Status 404 error, verify that the url you entered is consistent with the url specified under URL pattern in section 4.2. Note that the url is case sensitive.

6 Matlab examples

The full source code for the examples given in this chapter is available in the directory "matlabcode/examples". This applies to the trial version of the Webservice toolbox as well.

The Modelit Webservice Toolbox can be used in a variety of ways, but all applications come down to a front-end making a request followed by the backend responding to the request.

A number of design parameters define how a webservice is set up:

Design parameter	Use	Popular choice
Request method	The method that the webservice should use to extract data from a HTTP request. The Matlab Webservice Toolbox detects the request method, extracts the data using this method, and transfers the data to a Matlab callback function.	<ul style="list-style-type: none"> • HTTP GET • HTTP POST
Exchange format for input	The convention that is used to represent data in the input string. The Matlab callback function must decode the input string accordingly.	<ul style="list-style-type: none"> • HTTP GET string • XML • JSON • Custom format
Content type for output	The method that an internet browser should use to show the webservice response to the user.	<ul style="list-style-type: none"> • text/html • text/xml • application/pdf
Exchange format for output	The convention that is used to represent the output data.	<ul style="list-style-type: none"> • HTML • XML • JSON • Custom format

It would be abstract and unproductive to give a systematic description of all possible options. This manual is limited to a few examples, see the table below.

Front end	Backend (Matlab Webservice)	Result	Examples in
Internet browser	Generates HTML Code	View web pages that are generated on the fly from Matlab	Section 6.1, 6.3
Internet browser	Generates PDF code	View a dynamically created PDF file	Section 6.4

Remote application with XML interface including PHP and webpage	Generates XML code	Use results of Matlab in remote application or web page	Section 6.2
Remote Matlab application	Generates a serialized Matlab structure	Use computational result of webservice in Matlab application	Section 6.5

The possibilities however are endless and the reader is encouraged to experiment with new ways to utilize the toolbox.

6.1 HTML example

The code listing below constitutes the basic steps to initialize a MatlabServer object that listens to a specific port and invokes a callback when a request is made on that port. The basic steps are:

1. Create a MatlabServer object that listens to a specific port;
2. Set the callback to be executed when a request is made;
3. Start listening to requests.

```
% Create a server on the specified port, port should be a port specified
in
% the web.xml in the matlabserver directory in the Tomcat root
server = createMatlabServer(4444);

% Set callback to be executed when server receives a request from client
set(server, 'ServerInvokedCallback', @HTMLCallback);

% Start the webservice
server.startServer;

% Note: A compact alternative for the lines above is:
% server=startMatlabServer(4444,@HTMLCallback)
```

The code for this example is found in:
matlabcode\examples\startMatlabServer.m

To run this example, do the following:

1. Make sure that Tomcat is running;
2. If you have verified the Tomcat installation before any Matlab function was running (see section 4.6), make sure Tomcat is restarted before proceeding (simply "stop" and "start" Tomcat, see section 3.3) . This is because a previous timeout might prevent the webservice toolbox to probe the port where the timeout occurred;
3. Verify that "4444" is consistent with the settings in web.xml (see section 4.2);
4. Start a new Matlab session
5. Copy and paste the code above to the command line;
6. Open an internet browser and type:
<http://localhost:8080/matlabserver/myExample?arg1=1&arg2=2>

You should now see the following HTML page:



The HTML code for this page has been generated by the function "HTMLCallback ". The m-code for this function is included as an example in the toolbox. The code is also listed below. HTMLCallback generates a few lines of HTML code, and sets "content type" to "text/html" so that the browser interprets the return value as HTML code.

The general implementation of any MatlabServer callback function is:

1. Get the request properties from the ServerEvent . Use event2struct to convert the ServerEvent to a Matlab structure, see section 0;
2. Set the response properties of the ServerEvent and return the ServerEvent to the client by using the sendResponse function, see section 0.

```

function HTMLCallback(obj, event)
% HTMLCallback - Callback called by a MatlabServer object for generating a
% HTML response
%
% CALL:
%     HTMLCallback(obj, event)
%
% INPUT:
%     obj:
    
```

```
% handle to the MatlabServer object which triggered the callback
% event:
% ServerEvent with query and response information
%
% OUTPUT:
% No output
%
% Copyright 2008-2013 Modelit, www.modelit.nl

% It's good practice to always surround the callback with try catch
% and always return an answer to the client
try
    % Collect some request information
    S = event2struct(event);

    % Generate the response
    response = HTML(S);

    % Return response to the client
    sendResponse(event, response,...
        'contenttype', 'text/html') % The MIME type
catch me
    sendResponse(event, errorMessage(me.message),...
        'statusCode', 500,... % Statuscode 200: OK
        'contenttype', 'text/html') % The MIME type
end

%


---


function text = HTML(S)
% HTML - Make a simple HTML string
%
% CALL:
% text = HTML(S)

text = {'<html><center>'
    sprintf('<h1>%s</h1>',datestr(now))
    sprintf('<h2>HTTP: %s</h2>',S.RequestMethod)
    sprintf('<h2>Query: %s</h2>',S.RequestQueryString)
    sprintf('<h2>From: %s</h2>',S.RemoteAddr)
    '</center></html>'};
text = [text{:}];

%


---


function text = errorMessage(msg)
% errorMessage - Make an HTML error message
%
% CALL:
% text = errorMessage(msg)

text = {'<html><center>'
    sprintf('<h1 >%s</h1>',datestr(now))
    sprintf('<h2 style="color:red">%s</h2>',msg)
    '</center></html>'};
text = [text{:}];
```

The code for this example is found in:
matlabcode\examples\HTMLCallback.m

To stop the example from running (and be ready for the next example)
apply on the Matlab command line:


```
>> server.stopServer;
```

6.2 XML example

By replacing "HTMLCallback" with "XMLCallback" in the example in section 6.1 we can use the Matlab Webservice Toolbox to implement a webservice that returns XML data.

To run the example, apply on the command line:

```
>> server=startMatlabServer(4444,@XMLCallback)
```

If you view the result in an internet browser you should see:



The code for this example is shown below:

```
function XMLCallback(obj, event)
% XMLCallback - Callback called by a MatlabServer object for generating an
% XML response
%
% CALL:
%   XMLCallback(obj, event)
%
% INPUT:
%   obj:
%       handle to the MatlabServer object which triggered the callback
%   event:
%       ServerEvent with query and response information
%
% OUTPUT:
%   No output
%
% Copyright 2008-2012 Modelit, www.modelit.nl

try
% Collect some request information
S = event2struct(event);

% Return response to the client
sendResponse(event, response,...
    'contentType', 'text/xml') % The MIME type

catch me
    sendResponse(event, errorMessage(me.message),...
        'statusCode', 500,... % Statuscode 200: OK
        'contentType', 'text/xml') % The MIME type
end

%


---


function text = XML(S)
% XML - Make a simple XML
%
% CALL:
%   text = XML(S)

% Make a simple XML
text = sprintf('<?xml version="1.0"
?><response><time>%s</time><method>%s</method><from>%s</from></response>',...
    datestr(now), S.RequestMethod, S.RemoteAddr);

%


---


function text = errorMessage(msg)
% errorMessage - Make an HTML error message
%
% CALL:
%   text = errorMessage(msg)

text = sprintf('<?xml version="1.0"
?><response><error>%s</error></response>', msg);
```

The code for this example is found in:
matlabcode\examples\XMLCallback.m

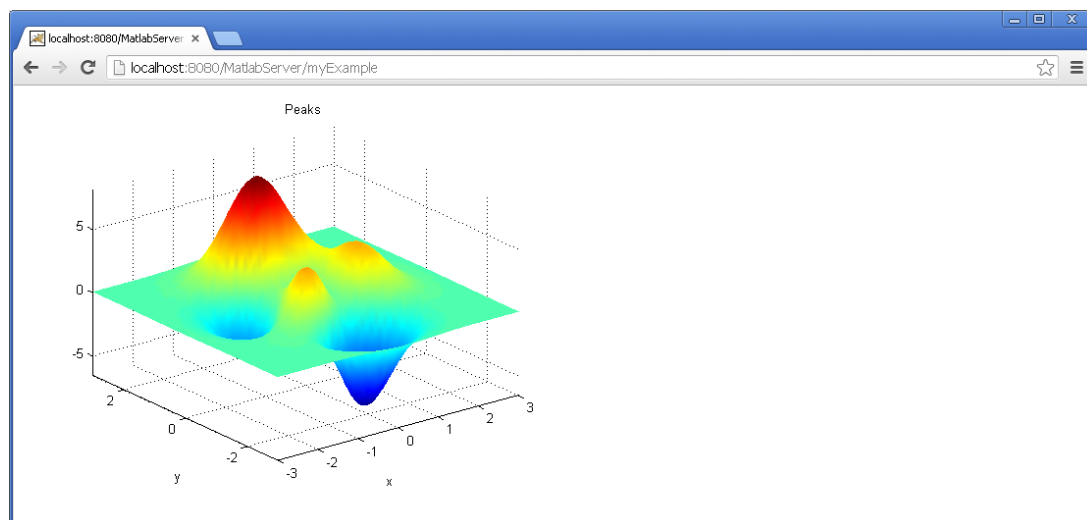
6.3 HTML example with graph

The Matlab Webservice Toolbox contains a number of utilities for converting Matlab figures into HTML code.

To view an example, activate the callback "FigureCallback" :

```
>> server=startMatlabServer(4444,@FigureCallback)
```

In an internet browser the output looks like:



The code of FigureCallback is included in the toolbox.

This is a simple example with a single graph. Obviously it is possible to generate complex html pages with multiple graphs, mixed with other html elements. One could even include hyperlinks or other controls in such a page that refer back to the Matlab webservice that created the page and thus create interactive pages.

```
function FigureCallback(obj, event)
% FigureCallback - Callback called by a MatlabServer object for generating a
% Matlab figure as response
%
% CALL:
%     FigureCallback(obj, event)
%
% INPUT:
%     obj:
%         handle to the MatlabServer object which triggered the callback
%     event:
%         ServerEvent with query and response information
%
% OUTPUT:
%     No output
%
%     Copyright 2008-2013 Modelit, www.modelit.nl
%
% It's good practice to always surround the callback with try catch
% and always return an answer to the client
try

    HWIN = findobj('tag',mfilename);

    if isempty(HWIN) || ~ishandle(HWIN)
        HWIN = figure('tag',mfilename);
        peaks;
        shading interp;
    end
end
```

```

end

% Generate the response
response = fig2html(HWIN);

% Return response to the client
sendResponse(event, response,...
    'contenttype', 'text/html') % The MIME type

catch me
    sendResponse(event, errorMessage(me.message),...
        'statusCode', 500,... % Statuscode 500: Interval Server Error
        'contenttype', 'text/html') % The MIME type
end

%
function text = errorMessage(msg)
% errorMessage - Make an HTML error message
%
% CALL:
% text = errorMessage(msg)

text = {'<html><center>'
    sprintf('<h1 >%s</h1>',datestr(now))
    sprintf('<h2 style="color:red">%s</h2>',msg)
    '</center></html>'};

text = [text{:}];

```

The code for this example is found in:
matlabcode\examples\FigureCallback.m

6.4 PDF example

One could use Matlab to create a PDF document on the fly or, as in the current example, read an existing PDF document from disk into a byte-array. If the content type is set to "application/pdf", browsers will display the data returned from the webservice as a PDF document.

To view an example, activate the callback "PDFCallback" :
>> server=startMatlabServer(4444,@PDFCallback)

In an internet browser the result is show as:



```
function PDFCallback(obj, event)
% PDFCallback - Callback called by a MatlabServer object for generating a
% .pdf file as response
%
% CALL:
%     PDFCallback(obj, event)
%
% INPUT:
%     obj:
%         handle to the MatlabServer object which triggered the callback
%     event:
%         ServerEvent with query and response information
%
% OUTPUT:
%     No output
%
% Copyright 2008-2013 Modelit, www.modelit.nl

try
% Generate the response
response = readBytesFromFile(fullfile(pwd, 'MatlabWebserver.pdf'));

% Return response to the client
sendResponse(event, response,...
    'contentType', 'application/pdf') % The MIME type
catch me
%In case of error, sent error message in HTML format
sendResponse(event, errorMessage(me.message),...
    'statusCode', 500,... % Statuscode 500: Interval Server Error
    'contentType', 'text/html') % The MIME type
end

%


---


function text = errorMessage(msg)
% errorMessage - Make an HTML error message
%
% CALL:
%     text = errorMessage(msg)
text = {'<html><center>'
    sprintf('<h1 >%s</h1>',datestr(now))
    sprintf('<h2 style="color:red">%s</h2>',msg)
    '</center></html>'};
text = [text{:}];
```

The code for this example is found in:
matlabcode\examples\PDFCallback.m

6.5 Communicate between Matlab sessions

The Modelit Webservice Toolbox makes it easy to call a Matlab function that runs on a remote server from any Matlab session and integrate the results of this call into the client program.

The current example presents the code for the utility "remote_eval" . This function is called in an equivalent way as the Matlab built in function "feval", but causes the called function to execute on a remote machine.

This mechanism opens all kind of useful options, like:

- Calling software modules that require datasets that are not available for the client computer because they are proprietary or very large;
- Protect proprietary software modules;
- Offload difficult computations to a powerful central CPU;
- Create a multiuser application;
- Centrally manage specific software modules.

The code of `remote_eval` is given below.

```
function value=remote_eval(url,fp,varargin)
% remote_eval - function that should be used on the client computer for
% remote execution of sepcified function
%
% INPUT
%   url
%       location of webservice. If empty, use example directory on
%       localhost
%   fp
%       function pointer for function to execute
%   varargin
%       arguments for function
%
% OUTPUT
%   value
%       result of function. Type depends on function.
if nargin==0
    %When called with no arguments: test this function.

    %Note that callback must be active in remote Matlab thread. Beware if
    %testing this example on localhost. Execute next line in SECOND Matlab
    %session.  server=startMatlabServer(4444,@MatlabCallback)

    %Use utility "remote_eval" to test
    %example 1
    value=remote_eval('',@max,[999,99,9],[9,99,999])
    %example 2 (catch error)
    value=remote_eval('',@max,[999,99,9],[9,99,999,9999])
    return
end

if isempty(url)
    url='http://localhost:8080/matlabserver/myExample?';
end

%Apply convention:
%- function and input data are stored in cell array
%- first element of cell array is pointer to function
%- next elements are input data
cmd_array=cat(2,{fp},varargin{:});

%serialize to byte string
cmdstr=serialize(cmd_array,true);

%call remote counter part
str=urlread([url,cmdstr(:)']);

%deserialize result
value=deserialize(str,true);

%verify that result does not originate from catch part
if isstruct(value)&&isfield(value,'identifier')
    rethrow(value);
end
```

The code for this example is found in:
matlabcode\examples\remote_eval.m

IMPORTANT:

The function `remote_eval` will only work if you install the webservice that replies to the requests in *another* Matlab session. Note the example will crash if you use `remote_eval` to call a webservice that is started in the same Matlab session.

To start the webservice that replies to `remote_eval`, start a Matlab session and execute:

```
>> server=startMatlabServer(4444,@MatlabCallback)
```

The listing of `MatlabCallback` is given below.

```
function MatlabCallback(obj,event)
% MatlabCallback - Callback called by a MatlabServer object for
% implementing a proxy
%
% CALL:
%   MatlabCallback(obj, event)
%
% INPUT:
%   obj:
%       handle to the MatlabServer object which triggered the callback
%   event:
%       ServerEvent with query and response information
%
% OUTPUT:
%   No output
%
% Copyright 2008-2013 Modelit, www.modelit.nl

try
% Get the request string
S = event2struct(event);

% Deserialize the request string
input=deserialize_v2(S.RequestQueryString,true);

% Generate the response.
% Note: input{1} is function to apply
%       inpu{2:end} are arguments to apply
output=feval(input{1},input{2:end});

% Return response to the client
sendResponse(event, serialize_v2(output,true));
catch
%In case of error, serialize error data
sendResponse(event, serialize_v2(lasterror,true));
end
```

The code for this example is found in:
matlabcode\examples\MatlabCallback.m

To run the example start a second Matlab session and execute:

```
>> value=remote_eval('',@max,[999,99,9],[9,99,999])
```

```
value =
```

```
    999    99    999
```


7 Using the Webservice Toolbox with compiled Matlab code

7.1 Reason for deployment in compiled mode

Typically the Matlab Webservice will be used to implement one or multiple webservices that run on a 24/7 basis, or to run multiple webservices in parallel for increased capacity.

To avoid that each webservice will permanently occupy a Matlab license, the Modelit Webservice Toolbox supports compiled Matlab code. In other words, any mfile that uses the Matlabserver Toolbox can be compiled to a standalone exe file. Moreover multiple instances of a standalone executable can run in parallel.

Compiling Matlab code to a standalone executable requires the Matlab Compiler™. The Matlab Compiler is a product of The Mathworks, see <http://www.mathworks.nl/products/compiler/>

7.2 First note: Setting java classpath in the deployment environment

In analogy to the Matlab development environment, the deployed environment requires a modified static java classpath as well. Like in the development environment this is done by placing a file classpath.txt (Matlab 2012a and before) or javaclasspath.txt (Matlab 2012b and later) in the directory where you execute your compiled executable.

The utility "verifyClasspathFile" facilitates the process. The utility checks the presence of the file "classpath.txt" or "javaclasspath.txt". If not present, the utility creates a template by copying the file classpath.txt from matlabroot/local. This template can then be modified and saved.

```
function ok=verifyClasspathFile
%verify if 'classpath.txt' or 'javaclasspath.txt' (depending on
%Matlabversion) is available

ok=true; %==> proceed with installing the webservice

%Define USECLASSPATH
% USECLASSPATH=true ==> file classpath.txt must be available in pwd
% USECLASSPATH=false ==> file javaclasspath.txt must be available in pwd
try
    if verLessThan('matlab','8.0')
        USECLASSPATH=true;
    else
        USECLASSPATH=false;
    end
catch
    %verLessThan does not exist until Matlab version 2008a
    USECLASSPATH=true;
end

if USECLASSPATH
    fname_required=fullfile(pwd,'classpath.txt');
else
```

```
    fname_required=fullfile(pwd,'javaclasspath.txt');
end

if exist(fname_required,'file')&&0
    %No message required
    return
end

%Copy classpath.txt from matlab root to template
fname_source=fullfile(matlabroot,'toolbox','local','classpath.txt');

[pth,nm]=fileparts(fname_required);
fname_template=fullfile(pth,[nm,'_TEMPLATE','.txt']);

%Create template file. This is a copy of classpath.txt in
%matlabroot/toolbox/local
copyfile(fname_source,fname_template);

%Display instructions for sysadmin
disp('No valid classpath file was found. ');
disp('This must be corrected before the webservice can be started. ');
disp('A copy of the system classpath file was saved to: ');
disp(fname_template);
disp('Expand this file with the path to MatlabServer.jar. Then Rename to: ');
disp(fname_required);
disp(' ');
disp('Press any key to exit program');
pause
ok=false; %==> stop with installing the webservice
```

The code for this utility is found in: examples\verifyClasspathFile.m

7.3 Second note: prevent the webservice from exiting

If a function that installs a webservice using the Modelit Webservice Toolbox is called from the Matlab command window (in other words: in the development environment), the webservice will remain active until Matlab is closed.

If the same function is compiled to a standalone executable and ran, the webservice will remain active until the function closes, which might be immediately after the webservice is posted. Obviously, this is not what we want, therefore some code must be added that prevents the function from closing.

This can be done by including an infinite loop after the webservice is posted as in the following code snippet:

```
%Start loop to prevent the program from returning to cmd window. Every 15
%minutes, display a status update in the console
while 1
    disp(sprintf('%s: #calls: %d #errors: %d Last call: %s',...
        datestr(now),N,E,datestr(TCALL)));
    pause(900) %wait 15 minutes
end
```

Another way might be to post a figure as in the code snippet below. Matlab will not exit before the user closes the figure.

```
HWIN=figure('Name','My Modelit webservice',...
    'Menubar','none', ...
    'closereq',@donotkill,...
    'Numbertitle','off',...
```

```

        'dock','on');
hmenu=uimenu(HWIN,'Label','Options'); %to activate docking
uimenu(hmenu,'Label','Close webservice','callb',{@removeWindow,HWIN});
%
function donotkill(obj,event)
warndlg({'Use menu "Close webservice" to stop the webservice'},...
        'Prevent accidental close','modal');
%
function removeWindow(obj,event,HWIN)
if strcmp(...
        questdlg('This will end the webservice. Continue?', ...
                'End application', ...
                'Yes', 'No', 'No'),...
        'Yes')
    delete(HWIN);
end

```

7.4 Code example

The code below combines the two notes listed above. The example may be compiled with the command:

```
mcc -m -C -d exe_webserver installwebservice
```

This results in an ".exe" and ".ctf" file. These two file may be placed in any directory of the machine that will run the webserver. On this machine the Matlab runtime library as found <http://nl.mathworks.com/products/compiler/mcr/> must be installed as well.

Also Tomcat Apache (see chapter 3 for installation instructions) and the MatlabServer.jar file (see chapter 4 for installation instructions) must be present on this machine.

```

function installwebservice
%entrypoint of webservice. This module will install the webservice.

%compile command (save results in directory "exe_webserver"):
%gcc -m -C -d exe_webserver installwebservice

%Assist end users with setting the classpath right
if ~verifyClasspathFile
    return
end

%Define some globals to keep track of the webservice status
global N %total number of calls
global E %total number of errors
global TCALL %instant of last call

N=0;
E=0;
TCALL=now;

% Create a server on the specified port, port should be a port specified in
% the web.xml in the matlabserver directory in the Tomcat root
server = createMatlabServer(4444);
% Set callback to be executed when server receives a request from client
set(server, 'ServerInvokedCallback', @webserviceCallback);
% Start the webservice
server.startServer;

%Start loop to prevent the program from returning to cmd window. Every 15
%minutes, display a status update in the console
while 1

```

```
disp(sprintf('%s: #calls: %d #errors: %d Last call: %s',...
    datestr(now),N,E,datestr(TCALL)));
pause(900)
end
%
%-----
function webserviceCallback(obj, event)
% webserviceCallback - Callback called by a MatlabServer object
%
% CALL:
%     webserviceCallback(obj, event)
%
% INPUT:
%     obj:
%         handle to the MatlabServer object which triggered the callback
%     event:
%         ServerEvent with query and response information
%
% OUTPUT:
%     No output
%
% Copyright 2008 - 2015 Modelit, www.modelit.nl

global N      %total number of calls
global E      %total number of errors
global TCALL  %instant of last call

% It's good practice to always surround the callback with try catch
% and always return an answer to the client
try
    N=N+1;
    TCALL=now;
    % Collect some request information
    S = event2struct(event);
    % Generate the response
    response = HTML(S);

    % Return response to the client
    sendResponse(event, response,...
        'contenttype','text/html') % The MIME type
catch me
    E=E+1;
    sendResponse(event, errorMessage(me.message),...
        'statusCode', 500,'contenttype','text/html') % The MIME type
end
%
%-----
function text = HTML(S)
% HTML - Make a simple HTML string
%
% CALL:
% text = HTML(S)

disp(S.RequestQueryString);

GETArguments = parseQueryString(S.RequestQueryString);
for k=1:length(GETArguments)
    input.(GETArguments(k).name)=GETArguments(k).value;
end

%Verify input arg "x" was specified
if ~isfield(input,'x')
    error('Input argument "x" is required'); %==> jump to catch
end

%Verify input arg "y" was specified
if ~isfield(input,'y')
    error('Input argument "y" is required'); %==> jump to catch
end

text = {'<html><center>'
    sprintf('<h1>%s</h1>',datestr(now))
    sprintf('<h2>HTTP: %s</h2>',S.RequestMethod)
    sprintf('<h2>Query: %s</h2>',S.RequestQueryString)
    sprintf('<h2>From: %s</h2>',S.RemoteAddr)
```

```

    sprintf('<h2>CPA: %s</h2>', ...
    num2str(addition(str2double(input.x), str2double(input.y)))
    '</center></html>');
text = [text{:}];
%
function text = errorMessage(msg)
% errorMessage - Make an HTML error message
%
% CALL:
% text = errorMessage(msg)
text = { '<html><center>'
    sprintf('<h1 >%s</h1>', datestr(now))
    sprintf('<h2 style="color:red">%s</h2>', msg)
    '</center></html>'};
text = [text{:}];
%
function result = addition(x,y)
result = x+y;

```

The code for this example is given in:
matlabcode\examples\installwebservice.m

8 Programming reference

All functionality of the Modelit Matlab Webservice is accessed through 2 objects: the MatlabServer object and the ServerEvent object. The is created by the toolbox function "createMatlabServer", while the ServerEventObject is passed on as an argument to any callback that is triggered by the by an internet request.

8.1 createMatlabServer

The MatlabServer Object listens to a port to requests from the MatlabServer servlet. If a request is made by the servlet a user defined callback can be triggered that handles the request and response.

Constructor:

Matlab code	obj= createMatlabServer(port)	
Input	port	A port number to which the object is listening. This argument must correspond to one of the ports specified with the "port" property in the web.xml script (see section 4.2).
Output	obj	The MatlabServer object that has been created.
Notes	Use createMatlabServer(port) to create a MatlabServer.	

Methods:

Matlab code	startServer(obj)	
Input	obj	The MatlabServer object.
Output	This command generates no output.	
Matlab code	stopServer(obj)	
Input	obj	The MatlabServer object.

Output	This command generates no output.
--------	-----------------------------------

Fields:

Usage: value = get(obj, fieldname) set(obj, fieldname, value)
port
The port number to which the object listens. This argument must correspond to one of the ports specified with the "port" property in the web.xml file (see section 4.2).
ServerInvokedCallback
A Matlab function handle. This function is called when a request is made on the port the MatlabServer is listening to. Like any Matlab callback, the callback is triggered with at least 2 arguments. Only the second of these, referred to as " ServerEvent object" is relevant in the present context.

Example Matlab code:

<pre>server= createMatlabServer(4444) set(server, 'ServerInvokedCallback', @myCallBack) server.startServer;</pre>

8.2 ServerEvent Object

The callback of the MatlabServer object is triggered when a request is made by the servlet. The callback is called with the standard Matlab callback arguments: An **object** and an **event**. For processing only this ServerEvent object is relevant. It contains the request information and can be used to set the response.

All interaction with the ServerEvent Object through the following methods

:

- getInputString
- event2struct
- sendResponse

Methods:

Matlab code	S = event2struct(event) Convert a ServerEvent to a Matlab structure		
Input	event	ServerEvent with fields:	
Output	S	RequestQueryString	Char string with the query string that is contained in the request URL after the path
		RequestMethod	This field is any of GET, POST, PUT, OPTIONS or DELETE Note: More information about HTTP methods can be found here .
		RemoteAddr	Char string containing the IP address from where the HTTP request was made
		RequestBody	Char string containing the body of the HTTP request. Note: The body is empty for HTTP get requests.
		RequestContentType	Char string with the MIME type of the body of the request. Note: More information about MIME methods can be found here .

Matlab code	<code>sendResponse(event, data, varargin)</code> Send response to client	
Input	<code>event</code>	The <code>nl.modelit.matlabserver.ServerEvent</code> .
	<code>data</code>	String or int8 array with data to send The input argument for this method is a string or an int8 array. Using the <code>nl.modelit.matlabserver.Utils.readBytesFromFile(file name)</code> and setting the correct Response MimeType (e.g. filename is a .pdf file and MimeType is "application/pdf" causes the webbrowser to show the a .pdf file.
	<code>varargin</code>	parameter value pairs, possible values: - statuscode (default 200) HTTP statuscode - contenttype (default 'text/html' data MIME type
	<code>status</code>	Number with the statuscode for this response. E.g. code 200 for OK. Note: More information about HTTP statuscodes look here .
	<code>contentType</code>	the <i>response</i> content type String with the MIME type of the response. Setting this value to for example 'text/html' or 'text/xml' enables the browser to correctly deal with the content. Note: More information about MIME methods can be found here .

Matlab code	<code>value = obj.getArg()</code>	
Input	<code>obj</code>	The <code>nl.modelit.matlabserver.ServerEvent</code> .
Output	<code>value</code>	<code>java.lang.String</code> ^{Fout! Bladwijzer niet gedefinieerd.} with the uery string or body (in case of a HTTP Post request).
Notes	This method is deprecated, use <code>getRequestQueryString</code> or <code>getRequestBody</code> instead.	

8.3 Troubleshooting

8.3.1 *Java exception*

The following error may occur when you try to start a MatlabServer twice within a single Matlab session.

```
java.net.BindException: Address already in use: JVM_Bind  
Exception in thread "Thread-18" java.lang.NullPointerException  
    at nl.modelit.matlabserver.MatlabServer$Caller.dolisten(MatlabServer.java:53)  
    at nl.modelit.matlabserver.MatlabServer$Caller.run(MatlabServer.java:36)
```

In this case you should restart the Matlab session.

8.4 **Callback cannot be reached for specific ports**

It is important that you first start the Matlab Webservice object(s) before you hit the monitor Tomcat button. Once Tomcat is active it attempts to forward incoming calls to the ports that were specified in web.xml (see section 4.2).

If Tomcat receives no answer from a specific port it will conclude that no process is listening to this port and will disable this port for the remainder of the session.

You will need to stop and start Tomcat to pick up any ports that have been disabled by the mechanism described above (see Figure 3).

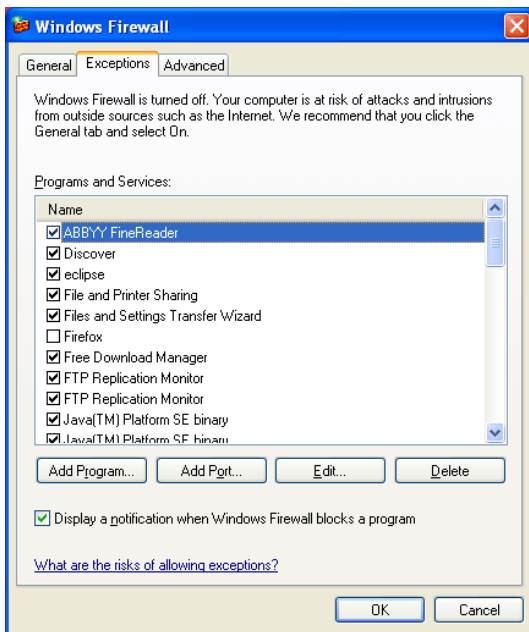
9 Windows Firewall Configuration on the Webservice Machine

9.1 Configuration on Windows XP

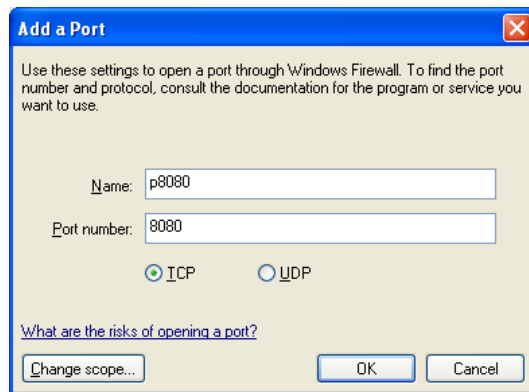
Below you will find screen dumps that illustrate how to configure the firewall on Windows XP Service Pack 2. For Windows 7 please refer to section 9.2.

9.1.1 Opening a port

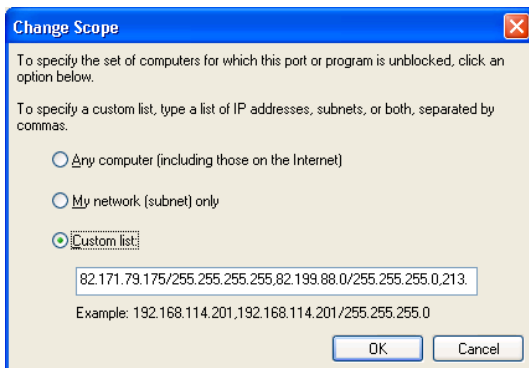
Go to the Control Panel and start the Windows Firewall dialogue. Use "Add Port" to allow messages to be passed on to a specific port. In our example we open port 8080.



Press <Add Port>



Specify (any) name
Specify port number
Press <Change Scope>



Specify a list of IP numbers that will have access to the webservice.

9.1.2 **Restricting access**

Enter all IP numbers. Note that these must be separated by with comma's and avoid typing blanks.

9.1.3 **Adding a range of IP-addresses (subnet mask)**

Example 1

Suppose you want to add any ip address that matches:

AAA.BBB.CCC.*

In this case add the following entry in the custom list

AAA.BBB.CCC.0/255.255.255.0

Example 2

Suppose you want to add any ip address that matches:

AAA.BBB.*.*

In this case add the following entry in the custom list

AAA.BBB.0.0/255.255.0.0

Note:

Once you have entered and saved a specific ip address such as:

AAA.BBB.CCC.DDD

Next time you open the dialog you may see:

AAA.BBB.CCC.DDD/255.255.255.255

This is the equivalent subnet mask.

9.1.4 **Active settings**

It is recommended to maintain a separate table of IP settings and users in your webservice deployment plan as well as a string to be entered in the XP firewall console.

IP address	User
123.123.123.0/255.255.255.0	All in company 1
124.124.124.124	Company 2

Complete string to be entered, note comma separation:

123.123.123.0/255.255.255.0,124.124.124.124

- Do not include any spaces in the string above.
- Note that windows may append "/255.255.255.255" if subnet mask is omitted.

9.2 Configuration on Windows 7

9.2.1 *Interactive setup*

The procedure to configure the firewall on Windows 7 involves the following steps:

- Go to windows Firewall;
- Go to Advanced settings;
- Add rule for incoming connections;
- Press "New rule";
- A window with 4 radio buttons opens;
- Select Port (Next);
- Select TCP and Specific local port (Next);
- Allow connection (Next);
- Check boxes Domain, Private and Public (Next);
- Specify Name (Complete);
- Now the new rule appears in the list;
- Select this rule and press "Properties";
- A tabbed window appears;
- Select the tab "Reach";
- Specify all external IP addresses that have access.

9.2.2 *Command line setup*

The interactive setup can be troublesome if many IP addresses are whitelisted, especially because the WIN 7 firewall requires one to specify all allowed whitelisted IP addresses one by one, and does not allow pasting a range of IP addresses, as is possible in the XP firewall interface.

A work around for this is to use the windows netsh command from a dos box.

- Open a command window
- Paste and execute the "netsh" command that is stored in a separate webservice deployment plan

An example if a "netsh" command is given below:

```
netsh advfirewall firewall add rule name=MYRULE dir=in localport=8080  
protocol=tcp action=allow remoteip=111.111.111.111,222.222.222.222
```

This command will add (not overwrite!) a rule name "MYRULE" and whitelist ip addresses 111.111.111.111 and 222.222.222.222 for port 8080.

10 Setting up the TCP/IP configuration on the server

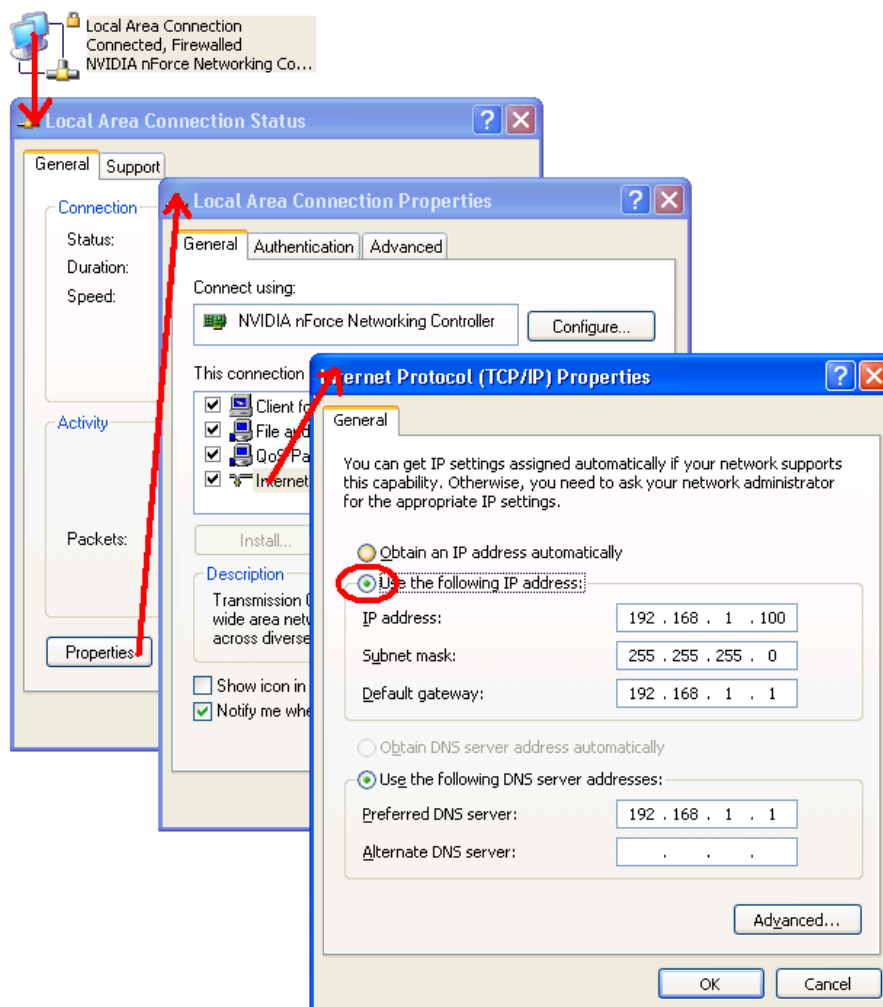
10.1 TCP/IP properties

To be able to set up port forwarding, it is required that the computer that runs the webservice, has a fixed IP address. This makes it possible to forward any request that comes in at the router to the server machine.

In some networks each machine gets its IP address automatically from the router in a range (for example) starting at 192.168.1.2. This mechanism is called DHCP.

It is recommended to fix the IP address of the webserver machine on an address that is outside the DHCP range. In our example we use the fixed address "192.168.1.100". This address must correspond with address configured in the port forwarding option, see chapter 3.4.1.

Note: If your router supports "bind IP to Mac Address" we recommend using this option (see section 11.2). In this case you may skip this chapter.



10.2 Verifying TCP/IP properties

Open a DOS prompt and enter the command "ipconfig -all".
Verify that IP address is equal to the address specified in the previous step.

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 5.2.3790]
(C) Copyright 1985-2003 Microsoft Corp.

C:\Documents and Settings\zijpp>ipconfig -all

Windows IP Configuration

    Host Name . . . . . : modelit-quad
    Primary Dns Suffix . . . . . :
    Node Type . . . . . : Unknown
    IP Routing Enabled. . . . . : No
    WINS Proxy Enabled. . . . . : No

Ethernet adapter Local Area Connection:

    Connection-specific DNS Suffix . :
    Description . . . . . : NVIDIA nForce Networking Controller
    Physical Address. . . . . :
    DHCP Enabled. . . . . : No
    IP Address. . . . . : 192.168.1.100
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.1.1
    DNS Servers . . . . . : 192.168.1.1

C:\Documents and Settings\zijpp>
```

11 Setting up port forwarding on the network router

11.1 Introduction

There are many network routers available on the market. Most of them have a console that can be reached through a specific IP-address in the local area network. As an example, this manual describes the procedure for a specific router. For other routers the procedure will be similar but not identical. The procedure may require some experimenting. If you experience any difficulties, you may search the Internet with keywords "port forwarding" or "port redirection" combined with the name of your hardware.

11.2 Step 1: disable DHCP on the server

It is recommended that DHCP is switched off on the machine that runs the Matlab Webservice (see chapter 10). Some routers support the function "Bind IP to Mac", in this case you can leave DHCP on the target machine switched "on" and assign a fixed IP address from the router console.

11.3 Example: setting up port forwarding on Draytek Vigor 2920

- Go to the web-console of your router. Usually this is located at: <http://192.168.1.1/>
- Add a new port redirection rule. Specify the following:
 - *Mode*: This is a Draytek specific option, select single;
 - *Service name*: any name may be specified here;
 - *Protocol*: select TCP
 - *WAN IP*: This is a Draytek specific option. Select All, Wan 1 or Wan 2. If you specify "All" your webservice will be reachable trough 2 ISP's. This allows users to apply failover: If the first IP does not respond, they can try the second one;
 - *Public Port*: select the port number that people will use to reach your webservice;
 - *Private IP*: specify the IP of the machine that runs the webservice;
 - *Private Port*: specify the port number that matches the parameter "Connector port" as specified in section 3.4.1. In the example "Public Port" matches "Private Port" but this is not required.

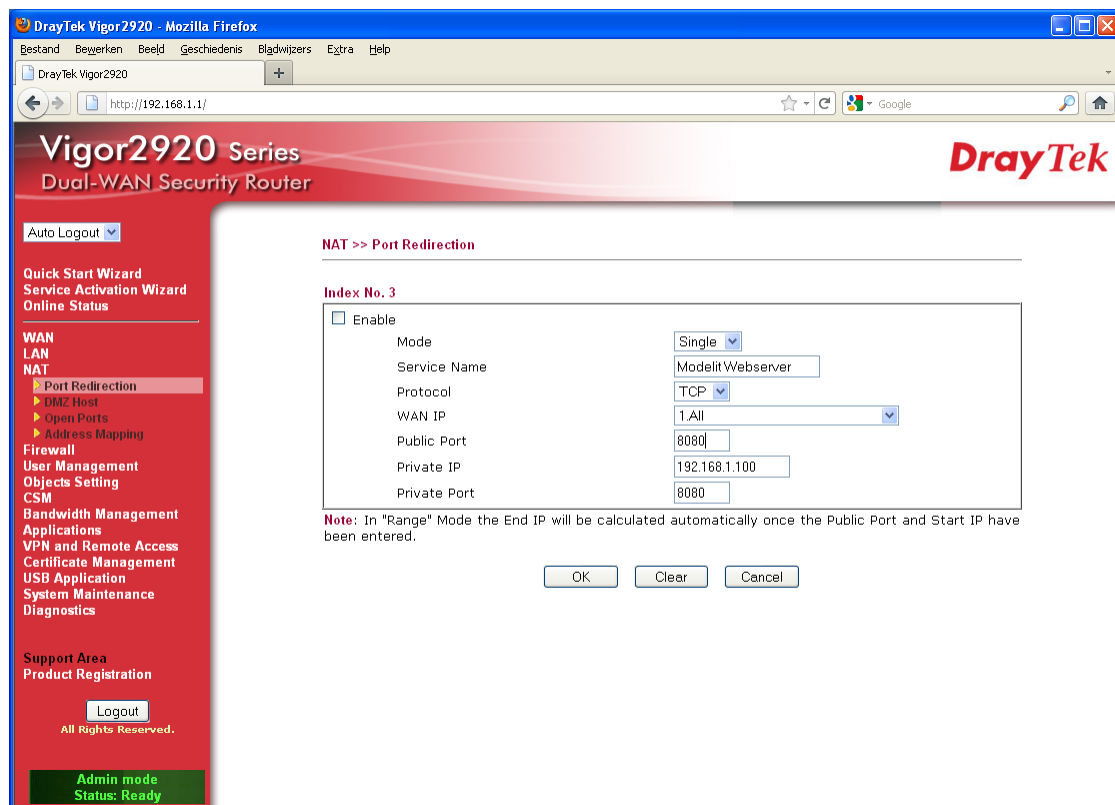


Figure 6: *Setting up Port Redirection on a Draytek router.*

11.4 Advanced topic: Setting up redundant servers and WAN's

If you need to meet rigid availability requirements, it might be a good idea to use redundant WAN connections and/or redundant servers. The latter also helps to increase capacity of the webservice.

Figure 7 shows an economical way to do this. For this you need:

- 2 independent ISP's, for example via cable and ADSL;
- a dual WAN router.

The configuration shown in Figure 7 makes ..

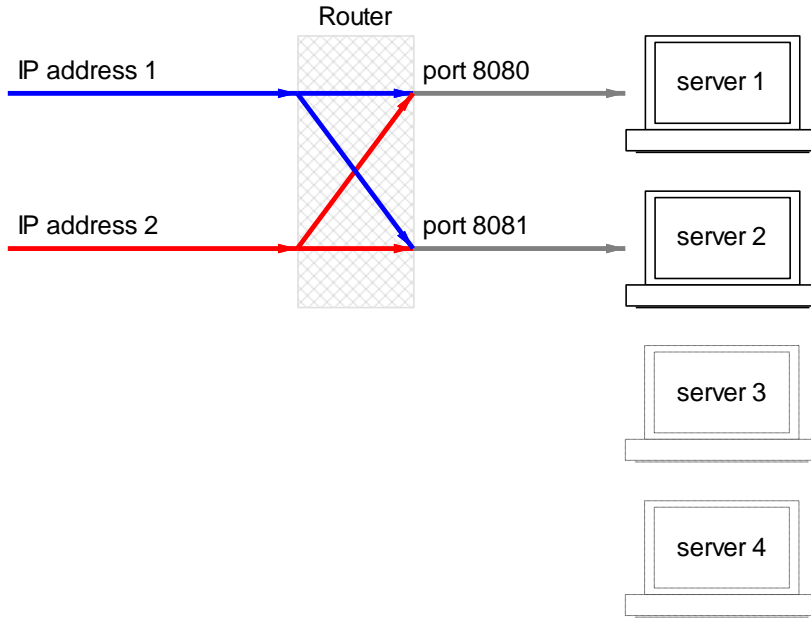
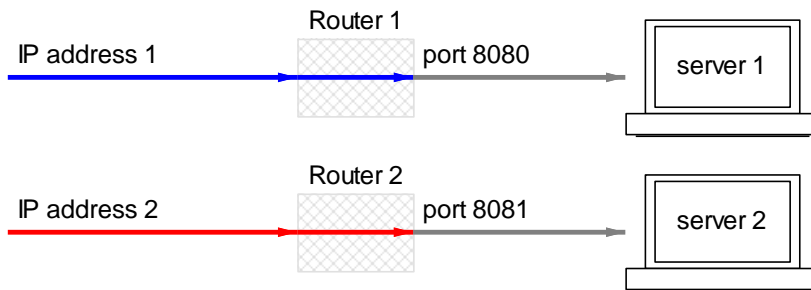


Figure 7: *Redundant WAN and server setup*

An alternative configuration would be to use two separate routers



12 Verifying the Matlab Webservice installation

12.1 Verification with a web browser

To call the webservice simply type the URL, port, directory and arguments of the webservice in the address bar of the web browser. For example:

```
http://localhost:8080/TripCast?id=123456&fromlat=52.66&fromlng=4.69  
&tolat=51.61&tolng=5.33&depart=1&time=20090607180000&reliability=0
```

To call the webservice from a remote location (a computer *outside* your local area network or a smartphone) simply type the URL, port, directory and arguments of the webservice. For example:

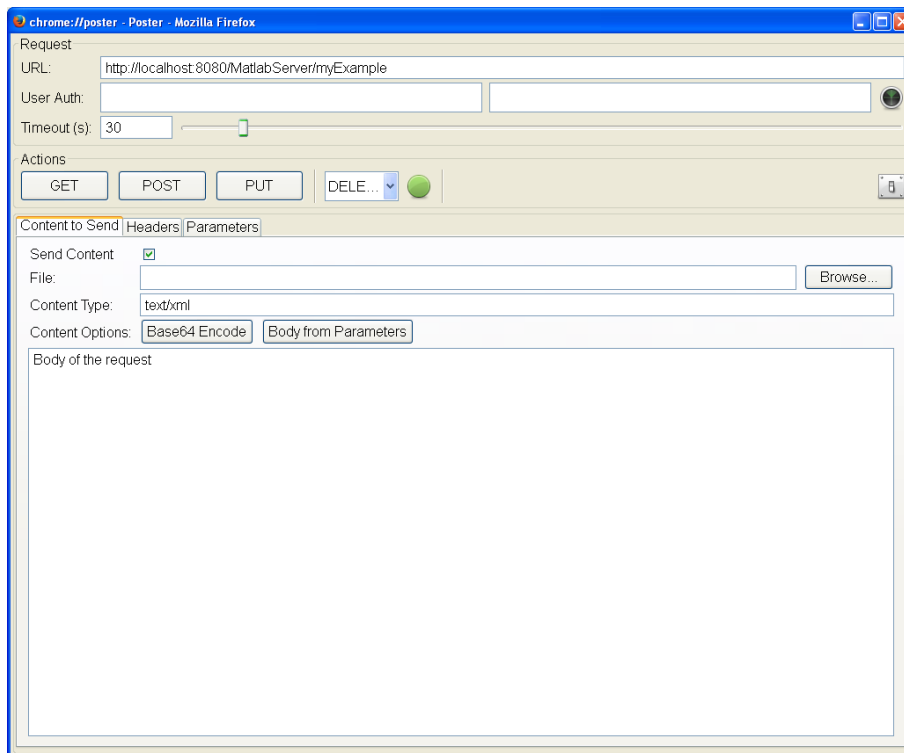
```
http://85.149.xxx.xxx:8080/TripCast?id=123456&fromlat=52.66&fromlng=4.69&tolat=51.61&tolng=5.33&depart=1&time=20090607180000&reliability=0
```

12.2 Verification with Firefox Poster

Firefox Poster is a developer tool for interacting with web that lets you make HTTP requests such as HTTP GET, HTTP POST and HTTP PUT. Poster is useful to test a webservice that handles HTTP POST request.

Poster can be installed in Firefox from:

<https://addons.mozilla.org/nl/firefox/addon/poster/>



12.3 Verification from Matlab

A deployed webservice can be called from Matlab by using the Matlab 'urlread' function. The function requires three arguments:

To invoke the webservice type:

```
str = urlread(url, method, args);
```

url	<p>A valid url with the location of the webservice e.g.</p> <pre>url = 'http://85.149.xxx.xxx:8080/TripCast';</pre> <p>or</p> <pre>url = 'localhost:8080/TripCast';</pre> <p>for a local webservice.</p>
method	<p>'post' or 'get'. The Webservice toolbox supports both methods, for more information on the difference between the two methods see for example: http://www.w3schools.com/TAGS/att_form_method.asp</p>
args	<p>Cell array with parameter/value pairs (all strings) e.g.</p> <pre>args = {'id','123456','fromlat','52.66','fromlng','4.69',... 'tolat','51.61','tolng','5.33','depart','1',... 'time','20090607180000','reliability','0'};</pre>

Note: The webservice can also be called by the adding the arguments directly to the url (after a '?') and invoking 'urlread' with one argument as follows:

```
str =
urlread('http://85.149.xxx.xxx:8080/TripCast?id=123456&depart=1');
```

However, this can cause problems if the arguments contain invalid url characters such as a 'space' or an 'ampersand &'. For a list of valid characters see for example:

http://en.wikipedia.org/wiki/Uniform_Resource Locator

12.4 Verification using proxy server

The text below assumes that you are testing a Matlab Webservice from a second computer.

To test a webservice one would typically need a computer *outside* the local area network that includes the computer that runs the webservice. All computers in a local area network share one external IP address. This

address shows up if one visits pages like <http://whatismyipaddress.com/>. If you make a call to the webservice that has been set up, this is the IP-address that you should use. However, if you make such a call from within the LAN, it is very likely that your router will not know what to do, and you will not reach your webservice.

A work around this is to use a so called proxy-site. These sites enable you to browse the internet while replacing your IP-address by that of the computer that provides the proxy. Obviously the first step is to find out this IP-address and to enable it in the firewall, again you may use a "whatsmyip" url for this purpose. You may find proxy sites by searching for "proxy sites".

13 Example: Integrating Matlab in web pages using XMLHttpRequest

This section contains a simple example in which the Modelit Matlab Webservice is used to generate an XML that is shown in a HTML webpage. All the required files can be found in the toolbox directory.

To run the example:

1. Start Tomcat (make sure the web.xml is properly configured see section 4)
2. In the Matlab command window type:

```
startMatlabServer(4444,@XMLCallback);
```

3. Double click on the file: hello.html to open it in a web browser. More information about the use of the XMLHttpRequest object can be found at http://www.w3schools.com/xml/xml_http.asp

This results in:



```
<!DOCTYPE html>
<html>
  <body>
    <div>
      <b>Time:</b><span id="time"></span>
      <br>
      <b>Method:</b><span id="method"></span>
      <br>
      <b>From:</b><span id="from"></span>
    </div>

    <script>
      // code for IE7+, Firefox, Chrome, Opera, Safari
      if (window.XMLHttpRequest) {
        xmlhttp = new XMLHttpRequest();
      } else {
        // code for IE6, IE5
        xmlhttp = new
ActiveXObject("Microsoft.XMLHTTP");
      }
      try {
        xmlhttp.open("GET",
"http://localhost:8080/MatlabServer/myExample", false);
      } catch(e) {
        alert(e);
      }
      xmlhttp.send();
      xmlDoc = xmlhttp.responseXML;

      document.getElementById("time").innerHTML =
xmlDoc.getElementsByTagName("time")[0].childNodes[0].nodeValue;
      document.getElementById("from").innerHTML =
xmlDoc.getElementsByTagName("from")[0].childNodes[0].nodeValue;
      document.getElementById("method").innerHTML =
xmlDoc.getElementsByTagName("method")[0].childNodes[0].nodeValue;
    </script>
  </body>
</html>
```