# Web Server Simulation System — User Guide — Version 1.0

Andrew Ferrier

June 20, 2002

## Contents

## 1 Licencing

## 2 An Overview of WS$^3$

WS$^3$ emulates a web serving system. It does this by reading in an XML (eXtensible Markup Language) input file that you specify, parsing it, and constructing a simulation of that system in memory that represents the system you have specified in the input file. It then executes that simulation system for a period of time that you have specified, using various parameters that you have specified. When the simulation is finished, WS$^3$ will output various data on the state of the simulation. Data can also optionally be output during the execution of the simulation.

There are three main objects which can exist in a WS$^3$ simulation system — clients, servers, and network nodes. There must be at least one client and one server, but network nodes are optional. Objects communicate by sending messages to each other along connections — these messages are either requests (sent by clients to servers) or replies (sent back from servers to clients in response to requests). Network nodes, if used, are used purely as intermediaries. Routes are used to specify ways of getting from one system object to another if they are not directly connected with a single connection.

Network nodes and servers have incoming message queues. Incoming messages are added to the queues and remain there until all the items ahead of them have been processed. For network nodes, processing merely involves sending on the message via an appropriate route. For servers, processing involves parsing the message (which should be a request), formulating a reply message and sending the reply message back via an appropriate route.

Clients do not have incoming message queues. Replies are processed immediately. Reply processing does not involve anything except recording the fact that the reply has been received.

## 3 System Requirements

- An operating system which can support the Java 2 Environment, such as Windows 2000 or Linux. Other operating systems also support the Java 2 Environment and WS$^3$ should operate correctly with those too, but it has not been tested with them. Installation instructions in this user guide are only given for Windows and Linux.

- A Java Run-Time Environment which is compatible with the Java Software Development Kit, version 1.3.1, as provided by Sun [2], such as the Java Run-Time Environment 1.3.1. *This environment must be correctly installed as per the instructions that come with it, such that the* PATH *and* CLASSPATH *environment variables are set correctly.*

  *Note:* If you want to build WS$^3$ from source code, you should have the full Java Development Kit installed rather than just a run-time environment.

- A Java XML Parser with the following features:

  - Support for XML 1.0 [3].
  - Support for DOM (Document Object Model) Level 2.
  - Support for JAXP (Java API for XML Parsing).

  An example of a parser with those features is the Xerces Java Parser, version 1.4.4 [1]. This has been tested with WS$^3$ far more extensively than any other Parser. Some other parsers may cause problems when used with WS$^3$. In particular, the Apache Crimson Parser has been briefly tested and I had some problems with it, as it does not support XML Schema validation.

  The Xerces Java Parser, version 1.4.4 is supplied with all pre-packaged versions of WS$^3$. It is subject to it's own licencing conditions, found in the file `LICENCE.xerces`. It is recommended that you use this version of Xerces as it has been tested with WS$^3$.

- If you wish to build WS$^3$ from the source code using the supplied makefile, you must have GNU (GNU's Not Unix) `make` [4] installed, as well as a standard suite of Unix utilities, especially `xargs`. On Linux or other Unix systems, this is often already installed. On Windows the easiest way to obtain this functionality is to install a Unix emulation layer such as Cygwin [5].

## 4 Installing WS$^3$

WS$^3$ will come distributed in one of the three following types of files:

- A `.zip` file.
- A `.tar.bz2` file.
- A `.tar.gz` file.

A `.zip` file should be used if you are planning to install on Windows. If you plan to install on Unix/Linux, use the `.tar.bz2` or `.tar.gz` file. Which you use will depend on what decompression programs are installed on your system — `gz` decompression is more widely available, but the `bz2` file will be smaller to download.

### 4.1 Installing on Unix/Linux

*Note:* The following commands should be entered from a shell prompt.

Uncompress the archive into a temporary directory, then change to that directory. For these instructions, we are going to assume that the directory is `/ws3`. For example, if using the `.tar.bz2` file[1]:

```
mkdir ~/ws3
tar xvjf ws3.tar.bz2 --directory=~/ws3
cd ~/ws3
```

Or if using the `.tar.gz` file:

```
mkdir ~/ws3
tar xvzf ws3.tar.gz --directory=~/ws3
cd ~/ws3
```

The archive is distributed with a pre-made binary `.jar` file that you can use to run WS$^3$ (after installing it). However, if you want to re-build WS$^3$, execute the following commands:

```
make clean
make all
```

In order to install WS$^3$, use the following command:

```
make install
```

### 4.2 Installing on Windows

Unzip the `.zip` file (archive) into a directory somewhere on your computer, for example `c:\ws3`.

### 4.3 Warning About Installation on any Platform

*Note: It is important to ensure* that however you install WS$^3$, the `xerces.jar` file is in the same directory as `ws3.jar`, *or* that `xerces.jar` is in your Java Runtime Environment's official extension directory. No other location will be likely to work, *even* if that location is in your `CLASSPATH` — this is because WS$^3$ is invoked with the `java -jar` option, which ignores the `CLASSPATH`. For more information, see [6]. The standard `Makefile` supplied with WS$^3$ will install both of these files in the same place, so this problem should not occur unless you try to install manually.

## 5 File Structure Once Installed

Once WS$^3$ is installed, the following directory and file structure will exist in the directory you have installed into[2]:

[1]The `j` option is not available on tar on some systems. Type `man tar` or `man bzip2` to find out how to decompress the file

[2]Only important files and directories are shown.

```
doc/        - The Java source files and compiled classes
tests/      - The input test files
ws3.jar     - The WS3 Jar file which can be used to invoke WS3
xerces.jar  - The supplied Apache Xerces Parser
ws3.xsd     - The WS3 XML schema
README      - Miscellaneous information
LICENCE.*   - Licence information
Makefile    - The makefile which can be used to rebuild WS3
```

## 6 Testing the Install of WS$^3$

### 6.1 Testing on Unix/Linux

Bring up a shell prompt and change to the directory where you extracted WS$^3$ (e.g. `~/ws3`). Execute the following command:

```
make test
```

### 6.2 Testing on Windows

Bring up a command prompt in Windows. From the directory where you extracted WS$^3$ (which, if you followed the instructions above, will be `c:\ws3`), and execute the following command:

```
make test_noinstall
```

WS$^3$ will build and install itself if necessary, then run through a variety of test simulation specifications designed to ensure that it is installed correctly. These test simulation specifications are the TST and CNC files referred to in the Testing and Evaluation chapters of the final report of this project [7]. Bear in mind that some of the tests are designed to fail!

### 6.3 Randomisation Test

WS$^3$ utilises the random number generator supplied with your Java Runtime Environment as a source of randomness. You can run a test to ensure that this random number generator does not 'repeat' itself too quickly. From the directory where you installed WS$^3$, type:

```
make randomtest
```

A utility will load and begin generating random numbers. It will take many years to complete execution, unless your random number generation implementation is faulty, in which case it may abort fairly quickly with a warning. You will want to abort it after a while, when you are confident than it has run for longer than you will ever want to use WS$^3$ for. Press Ctrl-C to do this.

See the final report for this project [7] for more information on these randomness issues.

## 7 Rebuilding the Javadoc Documentation

The WS$^3$ archive file is supplied with Javadoc documentation pre-built in the `docs/` directory. If you wish to rebuild this documentation, enter the following command from the directory where you unpacked the archive:

```
make doc
```

## 8 Creating an Input File

An input file for WS$^3$ is an XML file, which conforms to a certain XML Schema. This schema is supplied with WS$^3$ (the file supplied is called `ws3.xsd`), and specifies what elements can occur in the XML file, in what order, with what content, etc. — in other words, it constrains the content of the XML to a subset of valid XML. If you are not familiar with XML or XML schemas, it is recommended you use an XML editor which supports schemas to create any and all XML files which are input to the program, such as XML Spy [8]. WS$^3$ checks the XML file is valid for the schema[3], but the error messages thus produced directly reference the content of the schema and thus may be counter-intuitive to users not familiar with how schemas work. The XML file should conform to the basic layout shown in figure 1. This will ensure that the program correctly uses the right schema file, and is well-formed XML.

```
<?xml version="1.0"?>

<system xmlns="http://www.new-destiny.co.uk/andrew/project/"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        dataDumpPeriod="dataDumpPeriodInVirtualSeconds">
    <name>inputSystemName</name>
    <runtime>runtimeInVirtualSeconds</runtime>

    <!-- a number of system object elements -->

</system>
```

Figure 1: An outline layout for how an input XML file to WS$^3$ should look
.

### 8.1 Points to be Noted about Input Files

The following points should be noted about input files to WS$^3$:

- Identifiers are used to specify names and refer to system objects in the input file. Identifiers can contain only the alphanumeric characters (upper and lower case), and the underscore (_) and dash (-) characters, and must be between 1 and 127 characters long. All identifiers are case-sensitive.

[3]In most cases, assuming one is using a schema-validating XML parser library, such as Xerces 1.4.4.

Being case-sensitive means, for example, that `clientA`, `ClientA`, `CLIENTA`, and `Clienta` are all names of different system objects. However, one should try to avoid using different system objects which are differentiated only by case in that way, as it will most likely serve only to confuse, although WS$^3$ will understand the difference.

- All time units are generally assumed in this guide to be 'virtual seconds', or in terms of 'virtual seconds'. It actually makes no difference to WS$^3$ what one assumes the time units to be, but I generally refer to them as virtual seconds as that is a convenient time unit which seems appropriate in most cases. However, some aspects of the system will have 'sensible' default values if they are not specified in the input file, and these default values are based around the assumption that the virtual time units are seconds. Also, whatever time unit one uses, one *must* be consistent throughout an input file or WS$^3$ will not produce the desired results.

- Since a WS$^3$ input file is merely an XML file, XML-style comments can be used anywhere that they are normally valid in an XML file. Comments start with the string `<!--` and end with `-->` and they can span multiple lines. An example has already been used in figure 1 on the preceding page.

## 8.2 Input File Elements

A well-formed WS$^3$ XML input file consists of a single `<system>` element, with various attributes and subelements.

The `<system>` element *must* contain `xmlns` and `xmlns:xsi` attributes exactly as they are in figure 1 on the facing page. There are also various non-compulsory attributes of the `<system>` element:

- `defaultTTL` — The time-to-live value that is initially assigned to each message in the system. This is an integer greater than zero. The default value, if the attribute is not specified, is 32. See section 8.6 on page 14 for more information on time-to-live values.

- `dataDumpPeriod` — Specifies how often in virtual seconds WS$^3$ dumps out data. It should be an integer greater than zero. The default value, if the attribute is not specified, is 1. Data dumping is described in more detail in sub-section 9.1 on page 15.

- `traceLevel` — Specifies the level of tracing which WS$^3$ will use when running this simulation. This should be a integer which ranges between 0 (no tracing) and 4 (full tracing). A number higher than 4 will produce nothing extra beyond full tracing. A number lower than 0 will cause an error. See subsection 9.2 on page 15 for more information on tracing.

- `resetStatsPoint` — If present, specifies the virtual time point at which the summary statistics will be reset. This helps the user with obtaining accurate statistics about the execution, whilst compensating for equilibrium. For more information on this, see the section on 'Equilibrium' in the project final report [7]. Must be a number of virtual seconds greater than zero. If it is absent, or equal to, or less than, zero, the statistics will not be reset.

There are various sub-elements of the `<system>` element. Some are compulsory and some are optional. They are as follows, and whether they appear or not, *must* appear in this order:

- `<name>` — Specifies the name of input system. This name is not currently used for anything by WS$^3$ except as a title etc. when outputting information. It must appear once and only once.
- `<runtime>` — Specifies the total system runtime in virtual seconds. It has no default and must appear once and only once.
- `<client>`, `<networknode>`, `<server>` — These elements specify clients, network nodes, and servers in the simulation system. These three elements have various attributes and sub-elements. They are similar for each of three elements so they will be described together in sub-section 8.3. There must be at least one `<client>` element and one `<server>` element in the system — `<networknode>` elements are entirely optional.

## 8.3 System Objects

This sub-section describes attributes and subelements for the `<client>`, `<networknode>`, and `<server>` elements, collectively known as the system object elements.

There are various non-compulsory attributes for these system object elements. They are:

- `instances` — Specifies how many instances of a system object are created. This should be an integer greater than zero. The default value if the attribute is not specified is 1. If more than 1 instance is created, then WS$^3$ automatically creates multiple system objects with the same name and will represent those objects by appending `[`$n$`]` to the end of the name, where $n$ is the instance number of the system object (numbering starts from 0). When connections are created to an object, they should still be created just to the base name of that object, even if there are multiple instances of that object. Messages will be sent to a random object instance.
- `threads` — Valid only for the `<server>` system object. Specifies the number of concurrent threads operating on the modelled server. Must be an integer greater than zero. The default value if the attribute is not specifies is 1. If one sets this attribute, one normally wants to set the `processors` attribute as well.
- `processors` — Specifies the number of processors available on this server. The number of threads executing concurrently can never exceed this number. Must be an integer greater than zero. The default value if the attribute is not specifies is 1.
- `threadGrain` — Specifies the time grain which threads execute for. The length of time for which one thread can execute without giving up control to another thread can never exceed this value. Must be a decimal number of virtual seconds greater than zero. By default this attribute is set to 0.1 virtual seconds.

There are also various subelements for the system object elements. Some of them are compulsory and some are not. They are as follows, and must appear in this order if they appear:

- `<name>` — Specifies the name of the system object. This must comply with the identifier guidelines discussed in section 8.1 on page 6. This element must occur once and only once.

- `<connectto>`, `<routeto>` — These elements specify connections from, and routes for, the system object. At least one connection must be specified and zero or more routes can be specified. Connections and routes are explained in more detail in section 8.4 on the following page.

- `<creationDistribution>` — Valid only for the `<client>` element. Specifies the distribution used for creating the inter-generation times for client requests. The distribution is specified using a single distribution sub-element which is one of the elements described in section 8.5 on page 11.

- `<serviceTimeDistribution>` — Valid only for the `<networknode>` and `<server>` elements. Specifies the distribution used for creating the service times for network node queues and server queues. The distribution is specified using a single distribution sub-element which is one of the elements described in section 8.5 on page 11.

- `<destPossibility>` — Valid only for the `<client>` element. Specifies a server which it is valid for the client to create requests for. In other words, when a request is created by a client, WS$^3$ picks a random destination possibility from all those specified by that client (with equal probability). If only one destination possibility is specified for a client, the client always generates messages that are destined for that destination possibility. At least one `<destPossibility>` subelement must be specified for each `<client>` element.

  *Note:* The decision on which server to send a message to is made independently and before any decisions on how to route the message based on `<connectto>` and `<routeto>` elements (which are explained in more detail in section 8.4 on the next page).

- `<queueLength>` — Valid only for the `<networknode>` and `<server>` elements. Specify the length of the incoming message queue (not including the message currently being processed). Can be either a integer value greater than zero or the special value `infinite`, which ensures that queue is of infinite length. Obviously this is a modelling aspect which would not exist in the real world, but it is useful if one is unsure what the maximum queue length should be or one wishes to model a system in such a way that one can relate it to queueing theory easily. This element must appear once and only once.

- `<drop>` — Valid only for the `<networknode>` element. The value of the element must be a number between 0 and 1. Represents the probability that the network node should ‘drop’ any given message which it is asked to process.

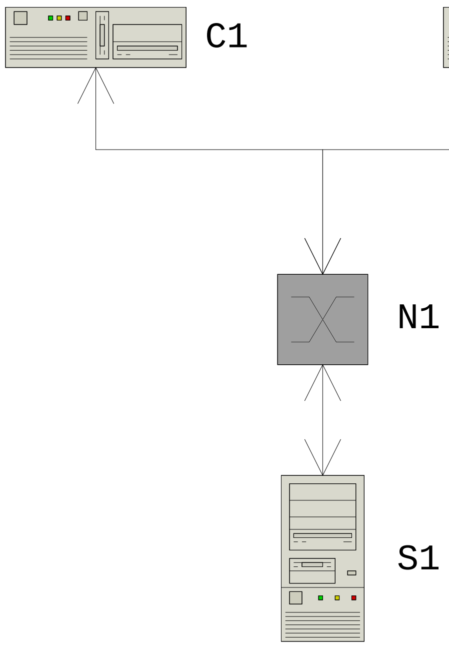

Figure 2: A simple example of a web client-server network

## 8.4 Connections and Routes

Each system object must have at least one outgoing 'connection' (although one can have as many as desired) and has zero or more outgoing 'routes'. A connection models a physical connection between two system objects — for example, if one were modelling a simple system with two PC (Personal Computer)s connected directly to the same hub on a simple LAN (Local Area Network), one acting as a web server (call it $S1$), and the other as a web client (call it $C1$), then one would probably model the link between these two with one 'connection' in WS$^3$ (though other models are possible which would be equally valid). Connections are *always* one-way, so in this case one would have two 'connection' objects — one attached to the client, going to the server, and one attached to the server, going to the client.

A route allows one to build more complex systems. If one had only two system objects, a route would be useless, but if there are more than two system objects, routes allow one to specify ways of getting from one system object to another, via another third party system object (or possibly more than one third party), even when there is no physical 'connection' between the two.

For example, say one extended the previous example so that there were two client hosts (call them $C1$ and $C2$) on the LAN, with the hub now modelled separately as a network node object ($N1$), and still with one server ($S1$). This example is shown in figure 2.

Ignoring the return journey from the server to the client to keep things simple, the way one would typically model the physical 'connections' for this in WS$^3$ is:

- There is a physical 'connection' from $C1$ to $N1$.

- Similarly, there is a physical 'connection' from $C2$ to $N1$.
- There is also a physical 'connection' from $N1$ to $S1$.

However, this is inadequate for WS$^3$ as it stands: if $C1$ generated a request to be sent to $S1$, WS$^3$ would not know the route it should take: all it knows are the targets of the *immediate* neighbours that it has *physical* 'connections' to. Hence we need to create some 'routes':

- Create a route at $C1$ where the destination is $S1$ and the route to take is via $N1$.
- Similarly, create a route at $C2$ where the destination is $S1$ and the route to take is via $N1$.

This would solve the problem: when $C1$ created a request to be sent to $S1$, for example, WS$^3$ would have a route whose ultimate target is $S1$, even though $C1$ does not have a physical connection to $S1$. Thus WS$^3$ would first sent the request to $N1$ where it would be queued on $N1$'s incoming queue. When $N1$ got round to dealing with the request, it would know how to send it to $S1$ because it has a physical connection to get it there. If there was no physical connection, $N1$ could sent the packet on again if it had a route which had the target $S1$ (although in our simple example it did not).

If this is confusing, think of connections and routes in the following way:

- A connection is attached to a source system object and has one parameter:
  - The name of the target system object.
- A route is attached to a source system object and has two parameters:
  - The *destination*, which is the *ultimate* target for messages.
  - The *route*, which is the name of an object the source object has a connection to — the next 'hop' which will bring the message 'closer' to the target.

Connections and routes are created between system objects by placing `<connectto>` and `<routeto>` elements inside the appropriate system objects in the input file. It is easiest to show this with an example (we will use the same setup as before, with a four-system-object network). The partial specification is shown in 3 on the next page.

## 8.5 Distribution Types

WS$^3$ supports a variety of statistical distributions which can be used for specifying client request generation, network node service time, and server service time. Each `<creationDistribution>` and `<serviceTimeDistribution>` element must contain a specification for one, and one only, distribution. One of the following distribution elements can be chosen:

- `<constant>` — This is a very simple distribution, that returns the same value every time it is used. For example, if one wants to model a client which generates a new request every 10 seconds, one would specify the `<creationDistribution>` of the client as in figure 4 on page 13.

```
<client>
    <name>C1</name>
    <connectto>N1</connectto>
    <routeto>
        <destination>S1</destination>
        <route>N1</route>
    </routeto>

    <!-- other specification elements -->

</client>
<client>
    <name>C2</name>
    <connectto>N1</connectto>
    <routeto>
        <destination>S1</destination>
        <route>N1</route>
    </routeto>

    <!-- other specification elements -->

</client>
<networknode>
    <name>N1</name>
    <connectto>S1</connectto>

    <!-- other specification elements: the network node needs
         no routes as it has a direct connection to S2 and
         we are not concerned about the return journey in
         this example -->

</networknode>
<server>
    <name>S1</name>

    <!-- other specification elements: we have specified no
         connection for the server as we are not
         concerned about the return journey in this
         example: but this would not be adequate for
         a real WS3 input file as every system
         object must have at least one outgoing
         connection -->

</server>
```

Figure 3: An example of how to use the `<connectto>` and `<routeto>` elements

```
<creationDistribution>
    <constant>10</constant>
</creationDistribution>
```

Figure 4: An example use of the `<constant>` distribution.

- `<exponential>` — Defines an exponential distribution, which follows the following formula:

  $$X = -\rho \times \ln r \tag{1}$$

  where $X$ is the variate, $\rho$ is the parameter, and $r$ is the randomly generated value.

  The single parameter is the average rate for the distribution. For example, if the XML in figure 5 were used in a `<client>` element, then the client would create 10 requests every virtual second.

```
<creationDistribution>
    <exponential>10</exponential>
</creationDistribution>
```

Figure 5: An example use of the `<exponential>` distribution.

- `<uniform>` — This element defines a uniform distribution, which returns values that are distributed uniformly over the interval specified. For example, if one wants a client to generate requests uniformly between very 3 seconds and every 5 seconds, one would specify the `<creationDistribution>` of the client as in figure 6.

```
<creationDistribution>
    <uniform>
        <lbound>3</lbound>
        <ubound>5</ubound>
    </uniform>
</creationDistribution>
```

Figure 6: An example use of the `<uniform>` distribution.

- `<geometric>`, `<erlang>`, `<positiveNormal>`, `<weibull>`, `<pareto>` — These distributions are all used in a very similar way to the distributions above. For the sake of conciseness, I have not detailed how each works, but they are all standard statistical distributions and there is plenty of information available for each one [9, 10, 11]. The Geometric and Pareto distributions only have one parameter each, which is included directly in the distribution element (similar to that in figure 4). The other three extra distributions use the parameters shown in table 1 on the following page.

| Distribution Name | Parameter 1 | Parameter 2 |
|---|---|---|
| Erlang | `<k>` | `<theta>` |
| Positive Normal | `<mu>` | `<sigma>` |
| Weibull | `<alpha>` | `<beta>` |

Table 1: Parameters for statistical distributions in WS$^3$

WS$^3$ does not actually support an 'instantaneous' distribution, where messages are routed instantly. However, one can be emulated, for most practical purposes, by creating a `<constant>` distribution and setting the routing time to be a value considerably smaller than any other time period used in the simulation specification. Bear in mind that although this works, it can slow down the simulation considerably.

## 8.6 Time-to-live Expiry

All messages have a TTL (Time-to-Live) value. For each message, it starts off at 32, unless you specify a different value for the `defaultTTL` attribute (which is attached to the `<system>` element). Each time a message is routed by a network node, the TTL value is decremented. If it reaches zero, then the message is discarded (though WS$^3$ produces a warning and the drop is recorded).

## 8.7 Constraint Enforcement

The XML Schema that is supplied with WS$^3$ enforces a number of constraints on the input files which WS$^3$ will accept. Most of these are common sense and they are fully specified in the XML Schema file itself, so they will not be fully detailed here. However, a quick summary of some of the more important ones follows:

- Item names must be consistent across the input file. For example, one cannot specify a `<connectto>` element which connects to a system object that does not exist.
- Item names must conform to the identifier guidelines found in subsection 8.1 on page 6.

# 9 Running WS$^3$

To run WS$^3$, simply issue the following command[4]:

```
java -jar ws3.jar inputfile.xml ../ws3.xsd
```

It is possible that this command may not execute correctly if another version of the Xerces parsing classes are installed on the system you are using, as is the case in DoC for example. In this case, you will have to execute WS$^3$ via the

[4]The `../` preceding the schema filename is necessary due to a bug in Java. You will always need to adjust this so that is points to the schema in the directory above the directory the schema is actually in.

compiled class file rather than via the `jar` file [5]. Execute the following command from the directory where you installed WS$^3$ [6]:

```
java -classpath .:xerces.jar doc.ajf98.websim.WebSim
    inputfile.xml ../ws3.xsd
```

In the above commands, replace `wsthree.jar` with the location on your system of the WS$^3$ `.jar` file. Replace `inputfile.xml` with the location of the desired XML input file specifying the system you wish to simulate, and `inputschema.xsd` with the location of where the supplied XML schema file is (typically the same directory as the `.jar` file) — but with `../` prefixing it.

The Java virtual machine will initialize, and load and run WS$^3$. WS$^3$ will execute, outputting extra information according to the options you specified in the XML input file, and then terminate. It may output a number of files, as explained in greater detail in section 10. If you want to abort the simulation whilst it is still executing, the key combination Ctrl-C should achieve this on most systems.

### 9.1 Data Dumping

WS$^3$ has the facility to dump data on the system status at certain user-specified intervals. These intervals are specified by the `dataDumpPeriod` attribute of the `<system>` element. This attribute specified how often data is dumped, in virtual seconds. The default is every 1 virtual seconds.

The data is dumped to a file which has a similar name and the same location as the input XML file, but has the string '`_dump.csv`' appended to the end. The name of this file is printed on the screen during the execution of WS$^3$.

### 9.2 Tracing

WS$^3$ has the facility to output a trace file during the execution of the simulation. This trace file describes actions that are occurring in the simulated system. It can help you to understand how the simulation works if you are unsure, and can also be useful if you are not getting the results you expect.

The trace file is created if the `traceLevel` attribute of the `<system>` element is greater than 0 (see sub-section 8.2 on page 7). The level of detail in the trace file is specified by that attribute. Experiment with the value of the attribute to get the level of detail you want.

The trace file is automatically created with a similar name and the same location as the input XML file, but has the string '`_trace.txt`' appended to the end. The name of this file is printed on the screen during the execution of WS$^3$.

## 10 Interpreting the Output of WS$^3$

WS$^3$ prints summary output on the screen once it is finished. This summary output should be self-explanatory to anyone familiar with simulation. It also

[5] This problem is due to a limitation in the way `jar` files work — most Java virtual machines ignore the `CLASSPATH` environment variable when executing a `jar` file.

[6] You may have to adjust the syntax of the command slightly for your system.

prints tracing output as explained in sub-section 9.2 on the preceding page, which should be equally self-explanatory. Data is dumped to a CSV (Comma Separated Variables) file (as explained in section 9.1 on the page before, and this data can be imported into spreadsheets or data analysis tools. Each row represents a discrete point in virtual time, with the left-hand-most column indicating what virtual time that was. Each column represents a measurable parameter of a system object at that point in time. The first row of the CSV contains headings for these columns which should also be self-explanatory.

For more information on how the output of WS$^3$ works, see the full final report for this project [7].

## 11 References

[1] Apache. *Xerces 1.1.4 Software.* A JAXP-compliant Java XML Parsing toolkit.
URL http://xml.apache.org/xerces-j/index.html

[2] *Sun's Java Page.*
URL http://java.sun.com/

[3] W3C. *XML (eXtensible Markup Language).*
URL http://www.w3.org/XML/

[4] Stallman, Richard and McGrath, Roland. *GNU Make.*
URL http://www.gnu.org/software/make/make.html

[5] *Cygwin.* A Unix-like environment for Windows. It can be used to compile WS$^3$ on Windows using the supplied makefile.
URL http://www.cygwin.com/

[6] *Download Extensions* (2002).
URL http://java.sun.com/docs/books/tutorial/ext/basics/download.html

[7] Ferrier, Andrew. *MEng Individual Project — Web Server Performance Simulation* (2002). The website for this project and the associated software (WS$^3$). Contains downloadable reports, program binaries and source code, and other related items.
URL http://www.new-destiny.co.uk/andrew/project/

[8] *XML Spy.* An XML editing suite.
URL http://www.xmlspy.com/

[9] Lewis, T.G. *Distribution Sampling for Computer Simulation.* Lexington Books, 1975. ISBN 0669971391.

[10] Field, Tony. *Simulation and Modelling* (2000). Lecture notes from Simulation and Modelling course, lectured by Tony Field, 1999/2000 academic year.
URL http://www.doc.ic.ac.uk/~ajf/Teaching/Simulation.html

[11] Banks, Jerry, Carson II, John S., Nelson, Barry L. and Nicol, David M. *Discrete-Event System Simulation.* Prentice Hall. ISBN 0130887021. A good, detailed, up-to-date book.

## 12 Glossary

**makefile** A makefile makes it easier to build a program or parts of a program. WS$^3$ is supplied with a makefile which uses GNU make [4].

*page 3*

**schema** A 'schema' in general is a set of constraints, or a layout, for a set of data. In the context of the this project and WS$^3$, a schema normally refers to an XML Schema.

*page 6*

**XML Schema** A 'schema' which constrains the content of an XML file.

*page 3*

## 13 List of Acronyms

| | |
|---|---|
| CSV | Comma Separated Variables  A record-based file where each new field is delimited by a comma, and each record is delimited by a new line. They can be used for exchanging information between spreadsheets and similar programs. |
| DOM | Document Object Model  An interface for accessing and updating documents, such as those written in XML. |
| GNU | GNU's Not Unix  A free software system. |
| JAXP | Java API for XML Parsing  A method for accessing and updating XML documents in the Java programming language. |
| LAN | Local Area Network  A network typically with no more than a few hundred hosts, distributed over a small (local) area, at most typically a building or two. |
| PC | Personal Computer  A small computer designed primarily for personal use. |
| TTL | Time-to-Live  A TTL value is normally used in network to avoid problems caused by network routing loops: a TTL value is decremented each time a packet or message is routed — if it reaches zero, the packet or message is discarded. This involves over-flooding of the network. WS$^3$ emulates TTL values for all messages. |
| WS$^3$ | Web Server Simulation System  The name of the software written for this project. The acronym WS$^3$ is sometimes written as WSSS. |
| WSSS | Web Server Simulation System  See WS$^3$. |
| XML | eXtensible Markup Language  A generalised markup language. See [3] for more information. |