ORBITER Multiplayer Project (OMP) V0.6.1

Friedrich Kastner-Masilko

February 28th 2012

1. INTRODUCTION

This project is intended to bring multiplayer features to the great Orbiter – a free space flight simulator created by Martin Schweiger, Ph.D. This is one of many attempts to accomplish the goal on both LANs and WANs and incorporates the idea of a single "universe" server.

Development is still ongoing by means of an open source project hosted on Bitbucket, a free collaboration platform that uses Git and Mercurial as version control systems. The project resides on http://bitbucket.org/face/omp. There is also a fork for the Orbiter Racing League project on http://bitbucket.org/face/orl-online. The later is considered the newest development at the time of writing this document.

2. OVERVIEW

The project started out as experimental proof of concept. The following functions were implemented in the first phase (pre-alpha):

- Basic networking functions (including session starting and joining, client management…)
- Communication protocol regarding vessel information exchange
- Synchronization mechanism
- Concept for the server-client-module approach to the problem of multiplayer interaction in Orbiter

The alpha releases had basic extensions:

- Inter-client-communication functions – chat functionality (even radio transmission delay simulation is possible)
- Trace mechanism to track vessels that are not in visible range
- A jump function to downsize simulation real time
- A simple orbital master system to provide cluster precision

The current beta phase is focusing on making those functions stable, and extending the project:

- Persistent vessel hosting (clients can leave the simulation and have their vessel propagated)
- Tunneling (connection possibilities for clients without full internet access)
- Flexible vessel creation and deletion
- Support for all vessel-specific recorded events (e.g. animations)
- Exchange of vessel-ownership
- Support for attachments and superstructures
- Space-time bubble concept to cope with time accelerations

For the next phases, possible extensions could be:

- Enhanced inter-client-communication functions (e.g. enhanced chat functionality, remote access functions, voicechat)
- MFD providing information about other client's vessels (i.e. the current trajectory of that vessels, etc.) and overhaul system information (e.g. server up time, user list, vessel state etc.). MFDs can act as chat interface, too.
- Vessel sharing, i.e.: many clients can control one vessel hosted by one of them. Huge vessels with many instrument panels can be created and controlled by a team in realistic scenarios.

3. DOCUMENT HISTORY

Version 0.1

September, 6th 2005 – Alpha release documentation.
September, 7th 2005 – Added Checklists.
September, 8th 2005 – Fixed typos and other errors.
September, 9th 2005 – Added single – LAN – SNTP server information.
September, 10th 2005 – Added jump feature
April, 17th 2006 – Second alpha release documentation.

Version 0.6.1

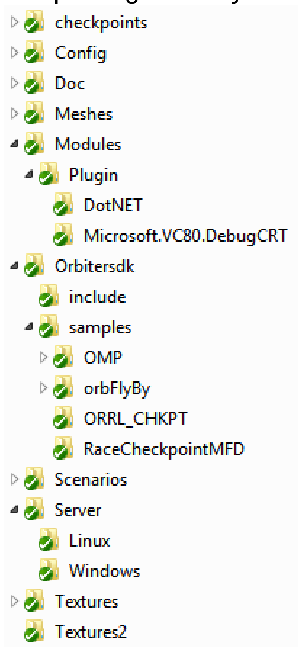
February, 28th 2012 – First beta release documentation.

4. INSTALLATION

Download the package from http://bitbucket.org/face/orl-online/get/0.6.1.zip into a temporary location. Extract the archive to a temporary folder, and copy the content of the single directory there (labeled with hash-code and name) into a vanilla Orbiter installation, overwriting files during the process. Be sure to enable the “use path information” feature of your favorite extraction tool.

These steps are not only tied to the above mentioned version 0.6.1. Instead, you can download any version as similar package via http://bitbucket.org/face/orl-online/get/<name_or_hash>.zip . In addition, it is recommended to use a vanilla install to reduce probability of addon-incompatibility. However, you can use any Orbiter install you like, or install additional addons in parallel to OMP or the ORL package.

The package usually has the following structure:



checkpoints	.. folder for checkpoint files and racing logs
Config	.. configuration files for ORL
Doc	.. documentation for ORL
Meshes	.. all necessary meshes for ORL
Modules	.. all Orbiter modules for ORL
Plugin	.. home of the RaceCheckpointMFD and OMPClient plugins
DotNET	.. .NET assemblies for OMP
Microsoft.VC80.DebugCRT	.. Microsoft debug runtimes for debugging OMP
Orbitersdk	.. complete project source code
include	.. changes to stock headers and/or additional libraries
samples	.. home of the ORL projects
OMP	.. OMP server and client sources
orbFlyBy	.. orbFlyBy sources – racing camera
ORRL_CHKPT	.. sources for checkpoint pylons
RaceCheckpointMFD	.. RaceCheckpointMFD sources
Scenarios	.. scenarios for ORL and default Ascension
Server	.. OMP Servers
Linux	.. for Linux – difference to Windows is in configuration and platform DLL
Windows	.. for Windows
Textures	.. textures for ORL
Textures2	.. high resolution textures for ORL

Note that the server files (executables and configurations) can reside anywhere on your system. In contrast to the previous releases, the configuration directory (/Config/) of an Orbiter installation is necessary now to provide the sol system information needed.

5. CONCEPT

The concept of OMP is a client-server-based architecture with time-stamped stream communication. The system uses SNTP timeservers to synchronize both clients and server to UTC. The 3 elements in the concept – client, server and timeserver – can be seen in Figure 1.

Not shown in this sketch is the communication between clients. This communication is similar to the UDP transmissions between client and server and resembles peer-to-peer networks in order to downsize streaming latency.

Basically, there is one server hosting a “universe” for all connected clients. This “universe” is a specific Orbiter configuration and will be negotiated with every client. I.e., every client must know with what Orbiter configuration it has to start the simulation session.

Clients start a TCP session with the appropriate server and can check out the server environment (users, information about vessels, scenario information etc.) similar to an IRC server. They actually join the multiplayer session by sending an appropriate command followed by their name, password and receiving port. The server responds with the reserved receiving port on the server side. The clients can then start the actual simulation link.

The TCP session is also used for transmission of text information of any kind to keep UDP packet sizes small. An open TCP session is therefore a must, if the right class (mesh) and name of the neighbour vessels are to be seen, because this text information is transmitted via TCP rather than UDP with an automated negotiation sequence.

As soon as a client joins the multiplayer session, it tries to synchronize the simulation's MJD to the appropriate space-time bubble's MJD (currently there is only one globally available server bubble). The server transmits this information by means of a "Keep Alive" UDP packet (KA). Concurrently, all vessels hosted by the client are transmitted to the server by means of the absolute position w.r.t. the appropriate gravitational body.

The server decides what vessel is in visible range of what other vessel and transmits the appropriate information by means of a "Group Information" UDP packet (GI). It consists of a target vessel ID (the vessel of the receiver client), a source vessel ID (the vessel of the sender client) and IP and port of the receiver client the sender should send to. Each client processes its GI list and sends either an absolute or a relative "State Information" UDP packet (SI) to the neighbour client. There are 5 reasons to send an absolute SI instead of a relative one:

1. The source vessel is not in the same space-time bubble – regarding space – as the target vessel. This is merely indicated by different gravitational bodies.
2. The target vessel is not available right now – if there was no absolute coordinates transmission before, the target vessel is not visible to the client yet, therefore it is not possible to calculate relative coordinates.
3. The source vessel velocity w.r.t. the gravitational body is smaller than 1000 m/s. This is considered "non-orbital speed" and should be within the tolerance of the established synchronization.
4. A jump command is running – check chapter 7. CLIENT .
5. The distance between source and target vessel is greater than the sub-visual range, or the velocity difference between source and target is greater than 1000 m/s. The sub-visual range d in meter is determined by:
   - the current simulation's viewport high h in pixels,
   - the current field of view fov in degree,
   - the maximum size of both vessels s in meter and
   - the formular $d < 0{,}5 \cdot \frac{s \cdot h}{\tan\left(\pi \cdot \frac{fov}{360}\right)}$. This estimates the user's eye-monitor distance with half a meter.

If a client receives an SI, it creates or updates the appropriate vessel with a generic name (based on the global ID of the vessel) and the default class (see Figure 2). This default vessel will be exchanged with the proper name and class later on, if the client was able to gather the information from the server via TCP. However, SI information is only taken into account if:

1. the reference object is available,
2. it is either relative to another vessel or absolute and the client is within the space-time bubble – regarding time – of the server and
3. the vessel to be updated is no superstructure slave.

If streaming of SI and GI is interrupted (either because the server is offline, the neighbour client is offline or – less dramatically – the vessel is out of visible range) for more than 5 seconds, the appropriate vessel is removed.

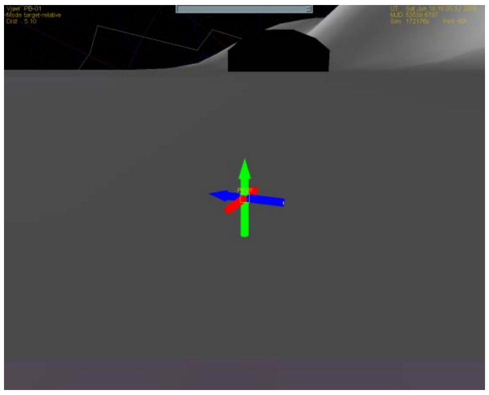


Figure 1 - OMP default vessel

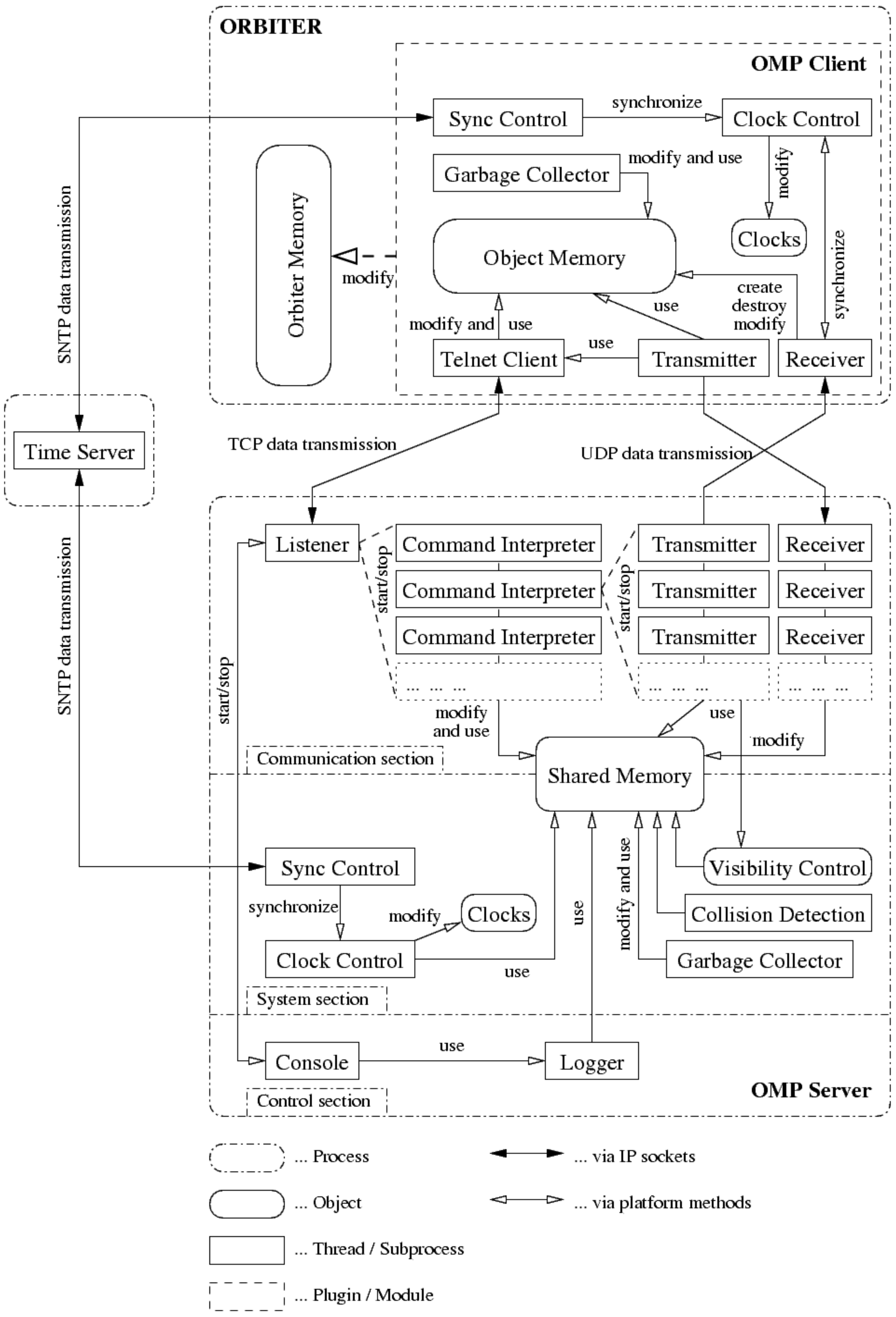


Figure 2 - OMP concept

6. SERVER

The main element in the OMP concept is the single server. The server is a Windows command line executable that can be run on any machine with Windows 2K, XP, Vista or Win7 operating system with .NET framework 2.0 installed, as well as on any POSIX machine with Mono framework 1.2.6 or higher installed. The executable uses a console configuration file, which location and name can be specified via command line argument.

A typical start command (on Windows) would be:

```
C:\Orbiter\Server\Windows>ServerConsole.exe console.xml
```

or, on POSIX machines

```
/opt/orbiter/Server/Linux>mono ServerConsole.exe console.xml
```

If no argument is given, “server.xml” (in the executable directory) is used as default server configuration file and “server.log4net” (also in the executable directory) is used as default logging configuration file. If the default files or the given console configuration file cannot be found, the server exits with an error message.

The console configuration file is in XML format and defines not only the location and name of both the server and the logging configuration files, but also additional settings for the command line server itself. The syntax is depicted in Figure 3, showing the XML scheme in graphical form.

The server configuration file is in XML format, too, defining all necessary parameters of the server. The syntax is depicted in 11. Appendix A. This file can be edited while the server is running. Changes will automatically be detected, and the server reloads the configuration.

The logging configuration file is again in XML format, providing a quite flexible logging facility through Apache’s log4net framework. Please check the framework’s project page at http:// http://logging.apache.org/log4net for details on the syntax, or consult the distributed server.log4net example files for Windows and Linux. The following loggers are used by the system (note: the character ‘#’ in names is a placeholder for whitespace):

`####`	General server and database log (info, warn, error and debug – the later prints unique ID collisions and/or misses)
`Chat`	Chat log (only info level)
`#TCP`	TCP subsystem log (info, error and debug – the later prints raw data with escape codes)
`#UDP`	UDP subsystem log (error and debug – the later prints raw data in hexadecimal format as well as caught exceptions)
`SNTP`	NTP subsystem log (info and error)
`VISI`	Visibility database log (only error level)
`#WEB`	Web server log (error only)

As soon as the server application has started, it begins to synchronize the local (virtual) clock with UTC via a simple NTP algorithm (not the one used in “real” NTP applications!). This algorithm works by means of pinging a random server from the specified set with SNTP requests in the defined sampling interval (typically 2s). If the server fails to reply, its fail count is increased. If the fail count hits a specified threshold, it is blocked and suspended for a defined amount of hit counts. A specified amount of samples is taken for each resynchronization step. After this step, the measured clock skew will be displayed together with some statistical information. Please don’t panic now… your systems clock will NOT be set to UTC, just an internal virtual clock is set properly.

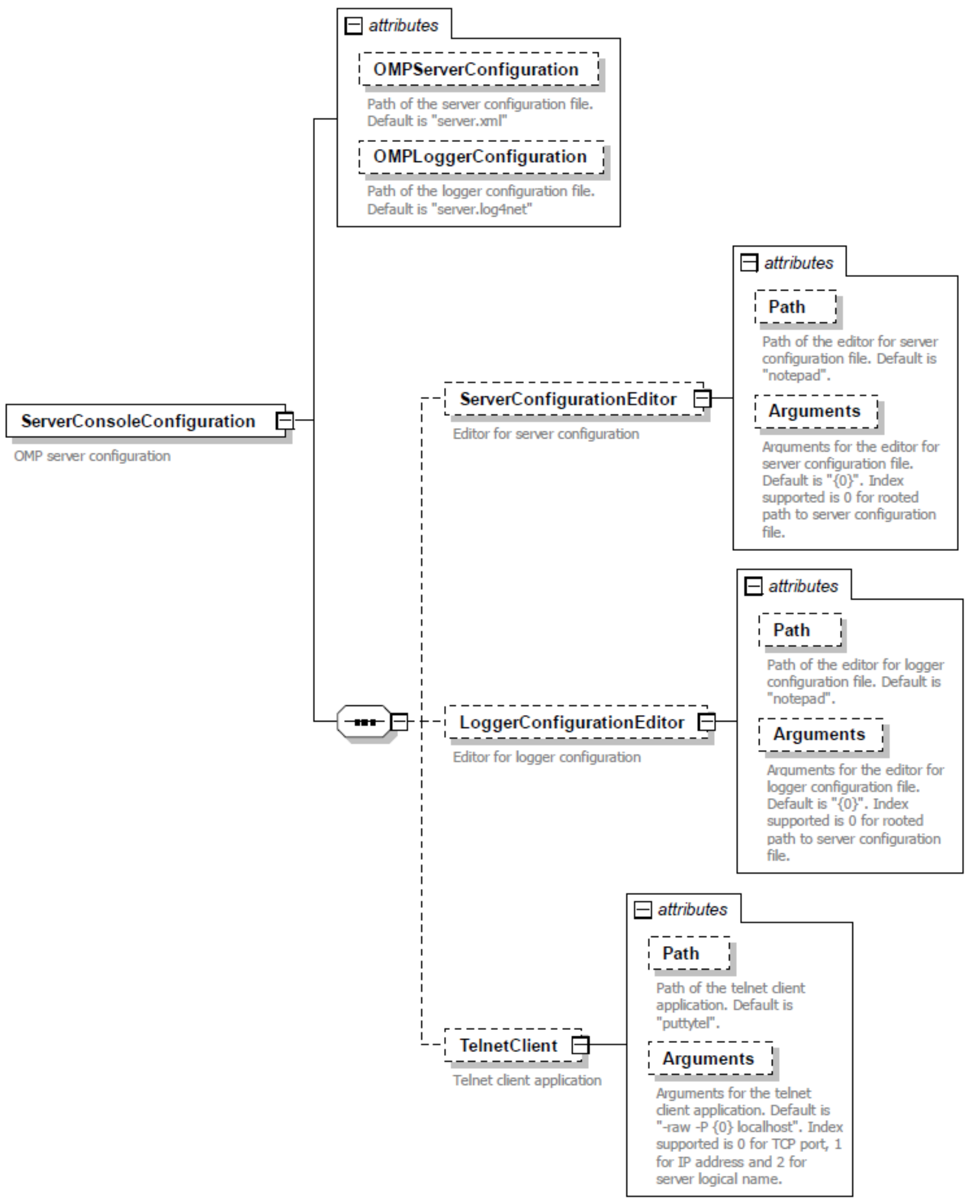

attributes
OMPServerConfiguration
Path of the server configuration file. Default is "server.xml"
OMPLoggerConfiguration
Path of the logger configuration file. Default is "server.log4net"
ServerConsoleConfiguration
OMP server configuration
attributes
Path
Path of the editor for server configuration file. Default is "notepad".
Arguments
Arguments for the editor for server configuration file. Default is "{0}". Index supported is 0 for rooted path to server configuration file.
ServerConfigurationEditor
Editor for server configuration
attributes
Path
Path of the editor for logger configuration file. Default is "notepad".
Arguments
Arguments for the editor for logger configuration file. Default is "{0}". Index supported is 0 for rooted path to server configuration file.
LoggerConfigurationEditor
Editor for logger configuration
attributes
Path
Path of the telnet client application. Default is "puttytel".
Arguments
Arguments for the telnet client application. Default is "-raw -P {0} localhost". Index supported is 0 for TCP port, 1 for IP address and 2 for server logical name.
TelnetClient
Telnet client application

Figure 3 - Console configuration file XML scheme

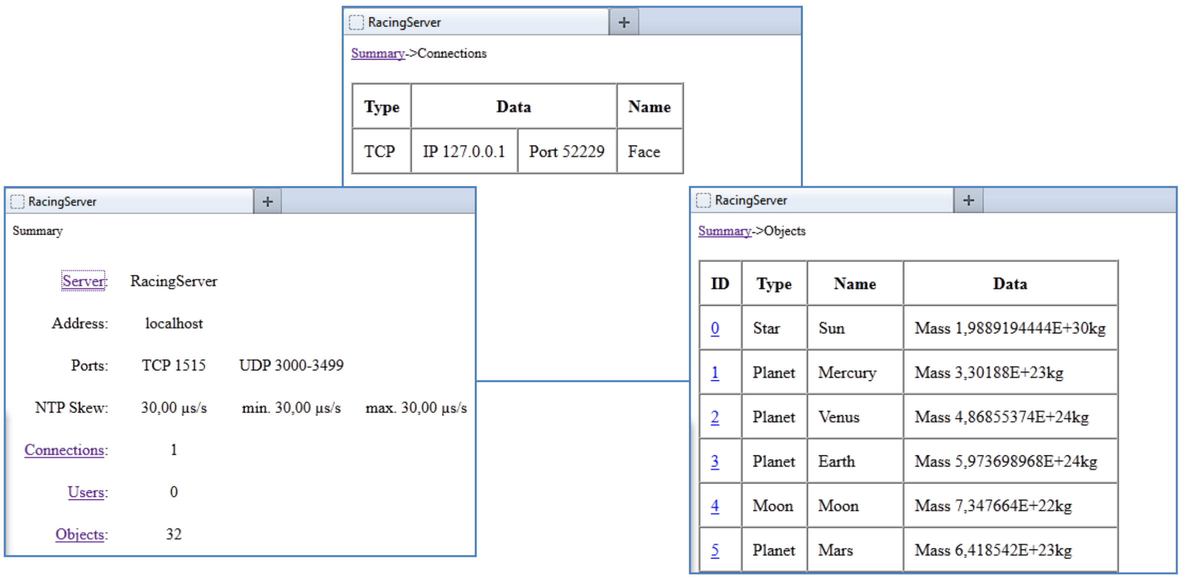
Concurrently, the server loads the specified solar system configuration from the defined Orbiter installation. All celestial objects are assigned a unique ID and loaded into a database that gets distributed to the clients.

The server then listens on the defined ports for client interaction. Usually, the command line server only logs information to screen and does not provide much interaction, but for maintenance purposes, there are some keys with special meanings. If you press these keys in the command line, the following happens:

C	Starts an editor to edit the server configuration file. This defaults to calling notepad.exe. Be advised that changes to the file cause the server to reload the configuration.
G	Triggers the .NET garbage collector to get rid of released IDs. Usually, this is not necessary, as the system does this automatically.
K	Toggles the server lock. If the server is locked, no new TCP connection can be made. This is useful in closed sessions to keep unwanted traffic out.

L	Starts an editor to edit the logging configuration file. This defaults to calling notepad.exe. Be advised that changes to the file cause the server to reload the logging configuration.
Q	Suspends screen logging and prompts to enter the administrator password to shutdown the server gracefully
S	Starts an administrator shell by means of the PuTTYtel application.

If you specify a web port in the server configuration file, the HTTP server is started. It lets you browse through the status of the server to check connections, users and objects. The pages served are rather spartanic in style, but easy to parse by both machines and people, as can be seen in Figure 4.



Summary->Connections

Type	Data		Name
TCP	IP 127.0.0.1	Port 52229	Face

Summary

Server:	RacingServer		
Address:	localhost		
Ports:	TCP 1515	UDP 3000-3499	
NTP Skew:	30,00 µs/s	min. 30,00 µs/s	max. 30,00 µs/s
Connections:	1		
Users:	0		
Objects:	32		

Summary->Objects

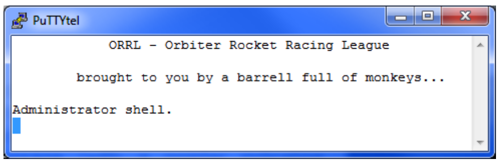
ID	Type	Name	Data
0	Star	Sun	Mass 1,9889194444E+30kg
1	Planet	Mercury	Mass 3,30188E+23kg
2	Planet	Venus	Mass 4,86855374E+24kg
3	Planet	Earth	Mass 5,973698968E+24kg
4	Moon	Moon	Mass 7,347664E+22kg
5	Planet	Mars	Mass 6,418542E+23kg

Figure 4 – Integrated web server

7. PROTOCOL

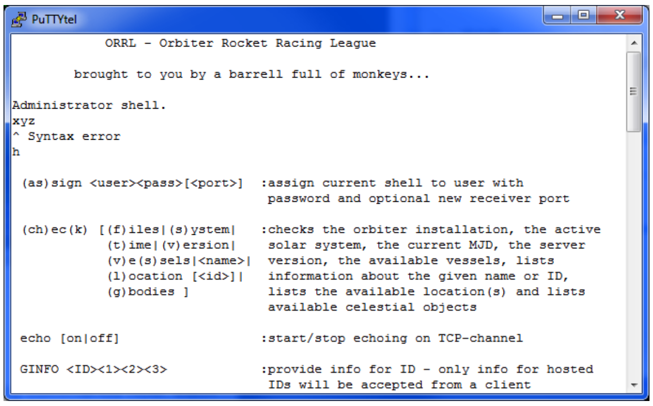
The communication between every client and the server starts out with the TCP session protocol. This protocol was designed to be human readable, both for debugging and extensibility purposes. Thus, the best way to explore the protocol is via Telnet.

You can connect to your server from any machine in the target network (even the machine running the server), or by means of using the standard telnet client that comes with the distribution (puttytel.exe) by pressing “S” in the server window. The server is prompting the specified welcome message and – if you use the server machine to connect – tells you that you are in an administrator shell.



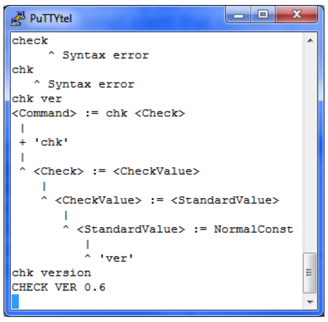
```
                ORRL - Orbiter Rocket Racing League

           brought to you by a barrell full of monkeys...

Administrator shell.
```

You can start entering commands now, each one finished by pressing <ENTER>. If you enter a wrong command, you will get a syntax error message (with a pointer indication where the error starts). The **help** command will show you a comprehensive list of all available commands.



```
            ORRL - Orbiter Rocket Racing League

        brought to you by a barrell full of monkeys...

Administrator shell.
xyz
^ Syntax error
h

 (as)sign <user><pass>[<port>]  :assign current shell to user with
                                 password and optional new receiver port

 (ch)ec(k)  [(f)iles|(s)ystem|   :checks the orbiter installation, the active
             (t)ime|(v)ersion|    solar system, the current MJD, the server
             (v)e(s)sels|<name>|  version, the available vessels, lists
             (l)ocation [<id>]|   information about the given name or ID,
             (g)bodies ]          lists the available location(s) and lists
                                  available celestial objects

 echo [on|off]                  :start/stop echoing on TCP-channel

 GINFO <ID><1><2><3>            :provide info for ID - only info for hosted
                                 IDs will be accepted from a client
```

If you enter a command, the command arguments are checked and a syntax error message is printed in the case of a missing or wrong argument. If the command is valid, it is processed and information is printed (e.g. checking the server version):



```
check
     ^ Syntax error
chk
   ^ Syntax error
chk ver
<Command> := chk <Check>
 |
 + 'chk'
 |
 ^ <Check> := <CheckValue>
    |
    ^ <CheckValue> := <StandardValue>
       |
       ^ <StandardValue> := NormalConst
          |
          ^ 'ver'
chk version
CHECK VER 0.6
```

Standard protocol commands:

(as)sign <user> <password> [<port>]	With this command, you can assign a shell to a specific user with the given password and optional port. It is automated and should not be used manually. However, this command can be used to obtain administrator privileges to remote control the server. For this purpose, give username “admin” and the administrator password. The server confirms your assignment by replying with an appropriate message.
ip [<address>]	This sets the IP address the current shell should be dedicated to. Every further action you take will be addressed to this IP. Without an address given, the current remote and set IP will be replied. The special IP “server” assigns the shell as “hair-pinning” client, providing support for clients inside the public server’s LAN. This is necessary whenever such clients have to connect to the public server using the private IP of it due to a NAT without hairpinning support.
echo [on | off]	This command switches echoing of characters on or off or shows the current setting without options given. This is useful if you use a telnet client without local echo or simply want to check back the text you send to the server. Echo off is default.
quit | exit	To stop the server connection gracefully, use this commands.
(j)oi(n) <user> <port> <password>	After sending this command, you join the server with the given user name and port – at which your client should listen for UDP packets – and secures it with the password. The server now confirms the joining by replying a port number the client should send SIs to – and immediately starts to send KAs to the given IP and port. If there already is a client connected with the current IP on the specified port, an error message will be replied. In addition, an invalid port number will be answered with an error message, too. A full server will also trigger an error. This is automated and should not be used manually. This command automatically assigns the shell to the newly joined port. If zero is used for the port, the server enters hole-punching mode and waits for the first client packet to arrive. It then sends back to the same port it received from (and with the same port it received it).

(l)ea(v)e <user> <password>	After sending this command, you leave the server. The user name and password authorizes your leave-request. Only authorized commands will be accepted, otherwise an error message will be returned. The server now confirms your leaving by replying an appropriate message – and immediately stops every activity to the IP and port. This is automated and should not be used manually. However, this command may be useful in the event of a failed connection procedure to eliminate zombies. This command automatically clears the shell's assignment.
(tr)ac(k) [off] [<ID>]	This command is especially important during a multiplayer session, because you can use it to track a vessel that is not in visible range. Since an OMP client only creates and displays vessels in visible range, it would be hard to launch to e.g. the ISS, because your MFDs cannot target a non-existing vessel. Therefore, the track command has been implemented. You specify the vessel you want to track by giving its ID. You can get this unique number for a specific vessel of a specific client by examining the reply of the **list** command. If the vessel exists, the server sends position information periodically to your client, but no information about name or class of the vessel, so you will only see the default vessel and a generic vessel name. But you can target this vessel with your instruments and plan maneuvers to get into visible range. As soon as you are in visible range to a tracked vessel, you will see 2 vessels in (almost) the same position. To switch off tracking, simply enter "off" together with the vessel ID. If you do not an ID or "off", the current tracking list will be displayed. You can add multiple trackings and delete all trackings by means of using "off" without any vessel ID. This command only works with a proper shell assignment. If your shell is not assigned to any port (either via joining or assigning), you will receive an error message.
(ch)ec(k) [(f)iles | (s)ystem | (t)ime | (v)ersion | (v)e(s)sels | <name> | (l)ocation [<ID>] | (g)bodies]	With this command, you can list the information for scenario files (DEPRECATED), the name of the server's solar system, its current MJD, its version, the supported vessel classes with mapping/redirections, information about a specific object name (not implemented yet), location information and the object database. All commandos are used during the automated startup procedure and should not be used manually – it should not do any harm to the system, though.
(l)i(s)t [(a)ll | (o)wn | (c)urrent | <name>] [(u)sers | (ob)jects] [(d)etailed]	As stated in the **track** command description, this command is useful to retrieve information about currently connected clients and their vessels. Depending on the options, clients will be listed with nick name, IP address and receiver port. Every vessel of a client will be listed with vessel ID, name and class name. There are currently 4 essential differences: 1. "all users" and "all users detailed" shows a nice list of all clients and their vessels, 2. "own users detailed" shows a list of all clients with the same IP address as the shell, 3. "own users" is like the detailed version, but suitable for automation (ugly to read) 4. everything else doesn't show anything. This command is prepared to be the information center for all kinds of information in later versions.

(h)elp	This command shows a list with descriptions of all available commands.
REQST <ID>	This command is used internally for vessel information transmission. Information for ID is requested. It is automated and should not be used manually – it should not do any harm to the system, though.
GINFO <ID> <1> <2> <3>	This command is used internally for vessel information transmission. It is automated and should not be used manually.
(t)alk:<message> | me:<message>	These commands provide the chat functions in the system. The message will be broadcasted to every online client's shell, prefixed with a "[<name or IP>]" in the case of **talk** and embedded in the brackets in the case of **me** (emoting). The default shell setting uses "t:" as prefix, so you don't have to type it every time you want to say something in the OMP client. Note that this string will be formatted with the prompt format at receiving clients' side.
RADIO:<message>	This command broadcasts the complete message to every client (including the RADIO prefix!). It is used by the internal radio signal delay simulation and should not be used manually.
(n)ic(k) [<name>]	You can set or clear the nickname used in the current shell with this command. If the nick is cleared, the IP is used as prefix.
(obs)erve [off] [<ID>]	Similar to the **track** command, this enables you to observe other vessels without them being in actual visible range to one of your vessels. Other than the **track** command, here the class and name information is exchanged and you'll get the state updates directly from the vessel's host. This command is intended to be used by mere observers that do not host vessels on their own, but just want to watch the session. In addition to the observed vessels itself, every vessel that is in visible range to it is automatically observed, too. Like with **track**, the "off" can be used to either selectively turn off an observation or the complete list. Without ID or "off", the command lists all current observations.
(pr)ompt <Format>	This command sets the prompt format of the shell. It supports .NET format specifiers, with index 0 ("{0}") replaced with the message and index 1 ("{1}") replaced with the date-time (based on the NTP algorithm's UTC) of the message. Check http://msdn.microsoft.com/en-us/library/aa720088%28v=VS.71%29.aspx for information on .NET format specifiers, or this nice blog entry: http://blogs.msdn.com/b/kathykam/archive/2006/03/29/564426.aspx.

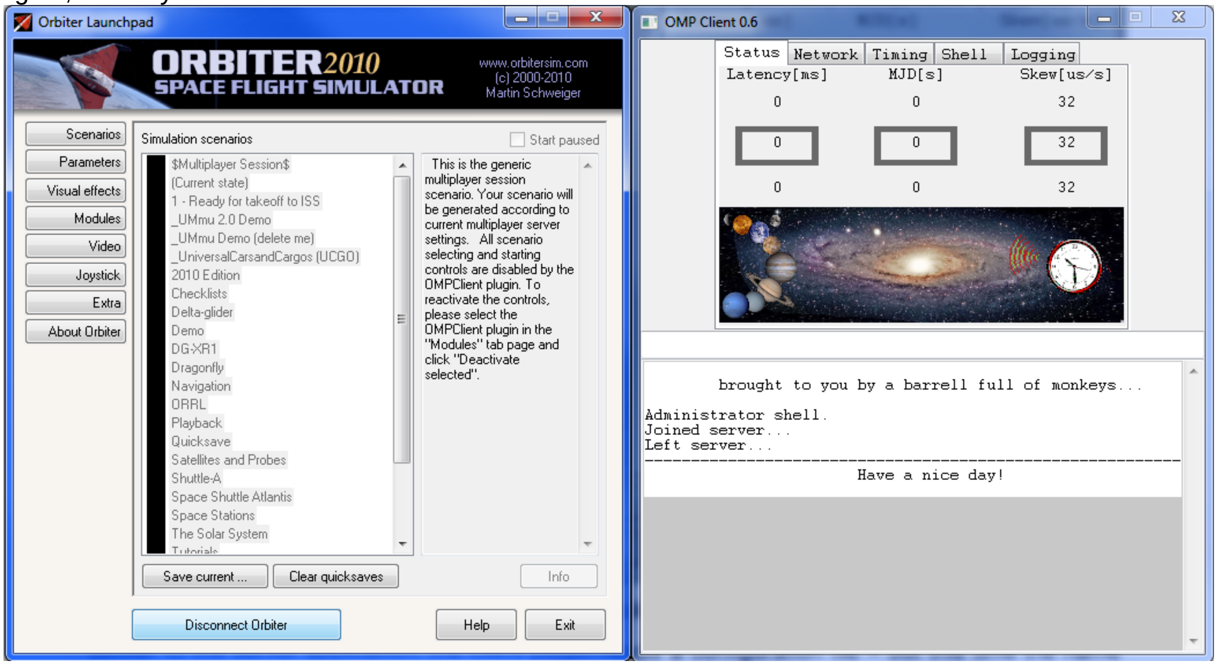
Administrator protocol commands:

log cut	OBSOLETE
log [<flags>]	OBSOLETE
sync reset | status | data	Currently only the status subcommand displays SNTP server information list. Both reset and data are obsolete.
(d)ump [(gl)obals | (c)o(n)nections | (cl)ients | ((l)og <chars> [last | <time>])]	Displays • global IDs, • connections or • clients as well as some statistics. Without any options, this command would perform a logging dump step, but all logging dumps are obsolete now with the log4net framework in place.
mjd (st)atus	Writes out a timestamp (local and UTC) as well as the current MJD.
mjd (r)eset <date> <time> <confirm>	Sets the MJD of the server to the specified date and time. As confirmation either “1”, “y” or “Y” will be accepted. If no confirmation was given, the system simply translates the date and time into MJD and prints it out. ATTENTION: Use quotes to ensure date (“YYYY.MM.DD”) and time (“HH:MM:SS”) is parsed correctly.
kick <ip>	Closes all TCP connections from the specified IP address.
kickban [off] [<ip>]	Kicks the specified IP and adds it to the ban list, or remove the IP from the list if used with "off" - no argument will display the list.

7. CLIENT

The client is an Orbiter plugin that can be activated in Orbiter's launchpad by going to the *Modules* tab page and checking the **OMPClient** entry within the *Multiplayer* section. You will immediately see the OMP client window in launchpad mode (see Figure 4). In this mode (indicated by the "universe" background in the status tab), there is no possibility to close the window by means of a button. The only way to close the window is closing Orbiter itself.

As soon as you activate the client, the first entry in the scenario selection control will be selected and the control itself will be disabled. Additionally, Orbiter's launch button turns to "Connect Orbiter" and the "Start paused" option will be disabled, too. This ensures that the user cannot start a wrong scenario for a given OMP session. If you deactivate the **OMPClient**, the controls will be enabled again, but only after a restart of Orbiter.



$Multiplayer Session$
(Current state)
1 - Ready for takeoff to ISS
_UMmu 2.0 Demo
_UMmu Demo (delete me)
_UniversalCarsandCargos (UCGO)
2010 Edition
Checklists
Delta-glider
Demo
DG-XR1
Dragonfly
Navigation
ORRL
Playback
Quicksave
Satellites and Probes
Shuttle-A
Space Shuttle Atlantis
Space Stations
The Solar System
Tutorials

This is the generic multiplayer session scenario. Your scenario will be generated according to current multiplayer server settings. All scenario selecting and starting controls are disabled by the OMPClient plugin. To reactivate the controls, please select the OMPClient plugin in the "Modules" tab page and click "Deactivate selected".

```
Latency[ms]      MJD[s]      Skew[us/s]
     0              0            32
     0              0            32
     0              0            32
```

```
            brought to you by a barrell full of monkeys...

Administrator shell.
Joined server...
Left server...
------------------------------------------------------------
                       Have a nice day!
```

Figure 5 - OMPClient module in launchpad mode

Similar to the server application, the client plugin needs a configuration file – but this time the name and location of the file is fixed: it must be in the same place as the plugin DLL and named *OMPClient.xml*. It is in XML format, with the syntax depicted in 12. Appendix B.

Like with the server, there is also a logging configuration file, again in XML format using the log4net framework. Its location and name is fixed to the same place as the plugin DLL and *OMPClient.log4net*. The following loggers are used by the system (note: the character '#' in names is a placeholder for whitespace):

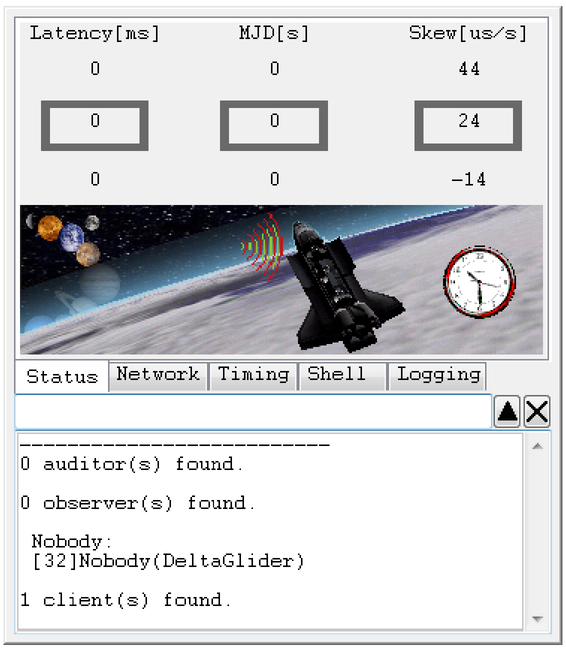
`####`	General client and database log (info, error and debug – the later prints various information about session status)
`Chat`	Chat log (only info level)
`#TCP`	TCP subsystem log (info, error and debug – the later prints raw data with escape codes)
`#UDP`	UDP subsystem log (info, error and debug – the later prints raw data in hexadecimal format)
`SNTP`	NTP subsystem log (info and error)
`#CLK`	Visibility database log (only error level)
`#WEB`	Web server log (error only)

Besides the launchpad mode, the client window can be in simulation mode, i.e., if you start an actual simulation and the render viewport opens (either in full-screen or windowed mode), the client window will switch to simulation mode and vice versa. These modes have some things in common, other things differ slightly:

- Both modes have the main layout elements *Settings tab control (tabs)*, *Command line (CL)* and *Text window (TW)*.
- Both modes have the tabs in the upper section, the CL in the middle and the TW at the bottom of the client window.
- In both modes, the tabs is fixed in width and height, the CL is always one line but stretchable and the TW is freely sizeable.
- In both modes, the client window has a minimum height and width – the tabs, the CL and one line of the TW must be visible.
- Both modes uses automatic tab splitting, i.e., if you make the client window broader, not the size of tabs changes, but you will have more tabs displayed concurrently as the space allows it.
- In launchpad mode, the tabs are at the top of the control, in simulation mode they are at the bottom. Don't ask why…
- In simulation mode, there are 2 additional buttons to the right of CL: one with a vertical arrow head on it, and one with an X. The later does what one would expect – it closes the client window. You can reopen the window via Orbiter's custom dialog feature. The arrow shifts the window in such a way, that the tabs are not visible anymore – additionally, the selected autohide mode will become active (see **Shell tab page** description).
- In launchpad mode, you can move and resize the window as you wish, in simulation mode, the window is fixed to the upper side and middle of the screen. If you resize the window in the later mode, the position will be maintained accordingly.

The next sections describe the layout elements in detail:

Status tab page:



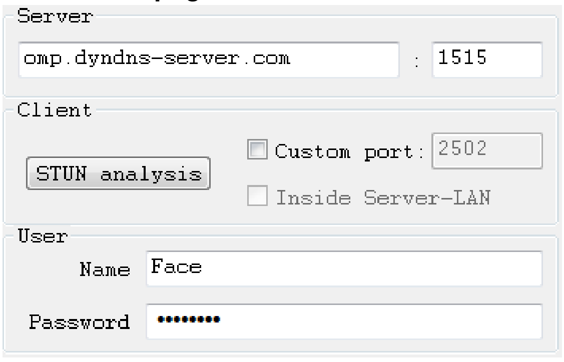
```
Latency[ms]     MJD[s]     Skew[us/s]
    0             0            44
    0             0            24
    0             0           -14
```

```
---------------------------
0 auditor(s) found.

0 observer(s) found.

 Nobody:
 [32]Nobody(DeltaGlider)

1 client(s) found.
```

This tab shows some statistics in the upper part. The numbers to the right are actual measurements of the maximum (upper), minimum (lower) and median (middle) value of the real-time clock skew w.r.t. UTC of the current machine. Normally, this is between 800 and 20 us/s in both directions. Additionally, the middle numbers show the maximum (upper), minimum (lower) and average (middle) value of the current machine's simulation time offset w.r.t. the server's simulation time. On the lower part of this tab, there are some animated icons that represent the current status of the system. Background of this "picture" is a universe if the program is in launchpad mode and a space shuttle on orbit in simulation mode. On the left side, the planets of the Sol system are displayed, if a TCP connection is active. On the right side, there will be:

- a solid clock icon, if everything's OK within the synchronization system,
- a fading clock icon, if an error occurred on an SNTP sampling point and
- no clock icon, if the synchronization stopped completely due to some fatal error.

Network tab page:



Server

omp.dyndns-server.com : 1515

Client

STUN analysis

☐ Custom port: 2502

☐ Inside Server-LAN

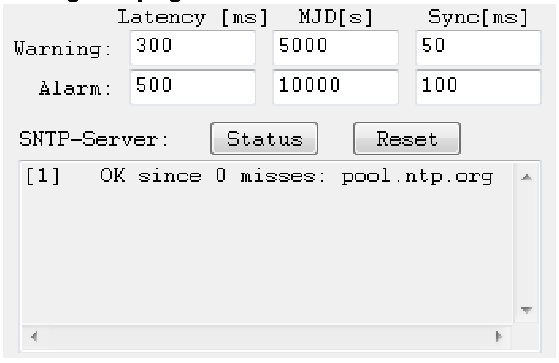
User

Name: Face

Password: ••••••••

On this tab you can enter the server's IP address as well as its receiving TCP port. User and password settings are important for the automatic startup procedure – do not leave them blank. For the analysis of the client connection to the internet, there is a button that initiates the STUN protocol. After pressing this button, it takes about 5 seconds for the results to appear in text form in TW. Without any checkboxes activated, the standard connection method uses a hole-punching technique to make the client receive UDP packets. However, this is not always successful, depending on your Internet access and relation to the server machine – the STUN analysis gives hints in this regard. If it is not working, the **Custom port** checkbox enables receiving through a specified port. If the client is in the same LAN as the server machine that hosts a public server, the **Server-LAN** checkbox can be used. With the later 2 options, the user has to setup a proper port-forwarding from the router to the client machine in order to receive UDP packets.

Timing tab page:



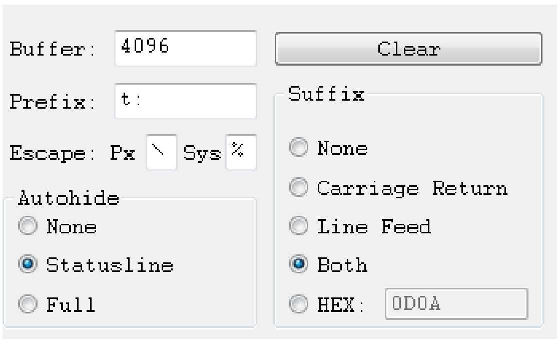
	Latency [ms]	MJD[s]	Sync[ms]
Warning:	300	5000	50
Alarm:	500	10000	100

SNTP-Server: Status Reset

```
[1]  OK since 0 misses: pool.ntp.org
```

In the upper part, none of the values do something useful yet. On the lower part, the button "Status" generates a report summarizing the status of the SNTP servers in use (BLOCKED means that there have been several errors connecting to this server). If your network connection is unreliable, or even broken, it may happen that all servers are blocked. The “Reset” button is reserved for this situation, but not implemented yet.

Shell tab page:



Buffer: 4096

Clear

Prefix: t:

Escape: Px \ Sys %

Autohide: None [✓ Statusline] Full

Suffix: None Carriage Return Line Feed [✓ Both] HEX: 0D0A

Here you can specify the shell buffer size, the suffix that will be concatenated behind your message, the prefix string that will be concatenated in front of your message (default is “t:” for sending normal entries as chat messages) and special characters for escaping prefix string (e.g. server commands) and escaping to client commands. In this tab you can set the auto-hide behavior as well. Whenever the arrow button (to the right of the shell input box) points down, the dialog box will hide according to this setting if the mouse leaves the dialog area, just like your windows taskbar do if you set it to auto-hide (see Figure 5). The "Clear" button clears the shell output window.

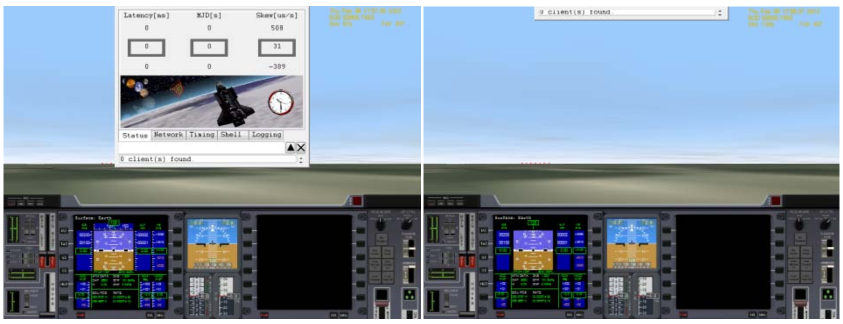
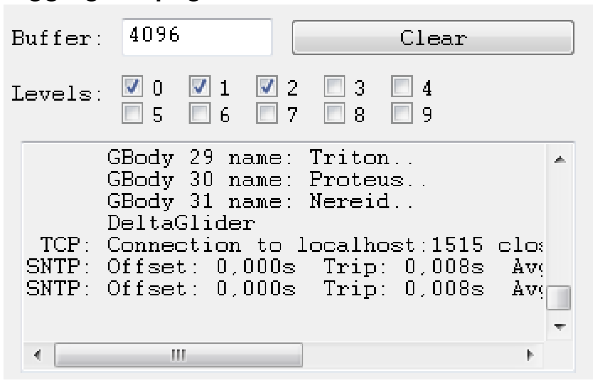


Figure 6 – Auto-hide feature

Logging tab page:


Buffer: 4096

Levels: [✓ 0] [✓ 1] [✓ 2] 3 4 5 6 7 8 9

```
      GBody 29 name: Triton..
      GBody 30 name: Proteus..
      GBody 31 name: Nereid..
      DeltaGlider
 TCP: Connection to localhost:1515 clo
SNTP: Offset: 0,000s  Trip: 0,008s  Av
SNTP: Offset: 0,000s  Trip: 0,008s  Av
```

This tab consists of a log output window and a textbox for buffer size setting, a button to clear the log and 10 checkboxes that represents the 10 built-in levels of logging. Right now there are 8 levels implemented:

0 .. Console messages
1 .. General messages
2 .. TCP messages
3 .. UDP messages
4 .. SNTP messages
5 .. DEBuG messages
6 .. CLocK messages
7 .. GarbageCollector messages

Every level can be switched on and off, respectively (default is 0,1 and 2 on, rest off).

Command Line:

If you switch focus to the client window via mouse movement, at first the CL gets the focus. You can enter text in the command line, delete it again and copy & paste text, as long as you do not enter a *Carriage Return*. As soon as you do that, the entered text will be passed to the internal parser as command. In addition, you can press up and down arrow keys to browse the history of previously entered escaped lines (server and/or client commands).

According to the content and the settings in the **Shell tab page**, the command can be interpreted as:

- text to be sent to the connected server via TCP or
- a shell command (default prefix is '%').

In the later case, the following commands are possible:

Keyword (case sensitiv)	Arguments (Type [Unit])	Description
Clock	None	Toggles the clock controller on and off. This way you can turn off time compression mechanism if you are connected to a server – highly experimental!
Echo	None	Toggles the local shell echo on and off – similar to the server side echo. It is off by default.
Irc	None	Toggles the automatic IRC reply (ping/pong mechanism) on and off. This is useful if you connect to an IRC server (some may remember…). It is off by default – OBSOLETE.
Event	None	Toggles the event transmission system on and off. It is off by default.
State	None	Displays information about internal settings.
Mjd	Integer [s]	Simulates a server MJD with the given offset to the current client MJD. Useful for tuning the PID algorithm if clock controller is off.
Kp	Float [1]	Sets the PID proportional factor.
Kd	Float [1]	Sets the PID integral factor.
Ki	Float [1]	Sets the PID differential factor.
Eps	Float [s]	Sets the PID tolerance epsilon. Below this threshold, the clients MJD is considered synchronized.

Delay	Integer [ms]	Sets the state information sender interval. This is the minimum time between packets sent out.
Block	Integer [ms]	Sets the state information sender block interval. This is the minimum time between packet blocks. A packet block
Idle	Integer [steps]	Sets the state information sender idle steps.
Rint	Integer [ms]	Sets the resynchronization interval – the time between each SNTP sequences for clock skew analysis.
Tdel	Integer [ms]	Sets the SNTP sample delay – the time between each SNTP sample within a resynchronization sequence.
Radio	None	Toggles radio transmission delay simulation on and off. This is a proof-of-concept feature and implements a client-side solution to the delay simulation problem. It is off by default.
Freq	Float [Hz]	Sets the radio transceiver. If delay simulation is on, a message will be only displayed at the proper time when it was sent at the currently selected frequency, just the same as it is in real life.
GCD	Integer [ms]	Sets the garbage collector sample time. This is NOT the timeout value for idle streams (fixed to 5 seconds for now), it's just the interval of garbage collector checks.
jump to	Integer [ID]	Jumps the focused vessel to the vessel specified by the given ID number. This must be a currently tracked vessel (see **track** server commando). After the jump, tracking will be switched off automatically.
Master	Integer [ID]	Sets the relative state vector master for the focused vessel to the specified ID number. This is useful in orbital clusters, were many clients want to see each other in the correct position. All clients must agree to a single master and set this value appropriately to maximize the cluster experience. No harm will be done to the system if this is not used in clusters, just relative positions can be wired. If the specified master is not in range, normal transmission will be used.
Version		Displays the version of the client.
List		Lists the index of all locally displayed vessels.
Add	Integer [ID]	Adds the specified locally displayed vessel to the server list.
Del	Integer [ID]	Deletes the specified locally displayed vessel from the server list.
Help		Displays the help information.

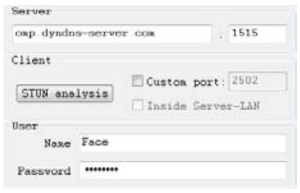
Text window:

The text window displays all received or echoed characters as they are - with two exceptions: every *Carriage Return* will be exchanged with *Carriage Return* followed by *Line Feed* – as this is the standard windows textbox line break – and every non-printable character will be displayed as pipe ('|'). The window supports automatic word wrap and can be scrolled (with the given buffer size as history).

Login:

In this version, the client performs all steps necessary for a proper OMP session by the means of an automated logon procedure:

1. The user provides the server address and port as well as the client connection options together with a user name and password (both must not be empty).



Server: omp.dyndns-server.com : 1515

Client: ☐ Custom port: 2502 ☐ Inside Server-LAN

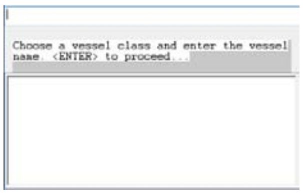
User: Name Face

Password ********

2. After clicking the connection button (the same as Orbiter’s launch button), the procedure connects to the server via TCP. If this fails, the procedure stops, but leaves the TCP connection running to let the user fix the problem manually, like he would do in a simple text client. I.e. you can still enter chat messages and examine the environment.

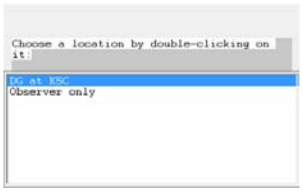
3. The procedure joins the user as a new client. In previous (and probably later) versions, it scanned the user list for an already connected session and used it when found. In this version, however, it is not necessary, as every disconnecting client will automatically leave, too.

4. The available vessel classes are then scanned and presented in a dialog for the user to choose from. To continue, press <ENTER> in the edit field.



Choose a vessel class and enter the vessel name. <ENTER> to proceed...

5. A scan for the available locations will be done and a dialog for the user to choose a location will be presented. To continue, double-click a location. If no locations are defined, the system will simply continue to load the single scenario definition of the server.



Choose a location by double-clicking on it:

DG at KSC

Observer only

6. A proper scenario with the correct MJD will be generated.

7. The Orbiter simulation starts fully automated.

8. The UDP transceiver starts and immediate MJD synchronization begins...

Logoff:

Logging off is automated, too:

1. If the user closes the simulation window, the procedure first stops the UDP transceiver.

2. It then stops the Orbiter simulation and returns to the launchpad.

3. At last, the OMP client window opens in launchpad mode and TCP connection stops.

8. LIMITATIONS

Since this is an alpha release, probability is high for bugs and pitfalls for both server and client. If you encounter any problems, please feel free to post them on the Orbiter forums or email it to me (face@snoopie.at). At this point I know of several problems, that I have not investigated far enough to provide a solution for:

- Massive multiplayer capabilities are still to be tested – I simply do not have the resources to do that on my own. It would be interesting how the system behaves with 50 or more orbinauts hanging around at the same time.
- Security and stability – at the moment there is only little security available. Since many features are still experimental (e.g. event hooking and forwarding, animation support, etc.), it is no exception to see a CTD with the client running.
- Threading concerns – because Orbiter seems to be a single threaded application, but the client plugin is using multithreading where possible, there may be race conditions sometimes somewhere…

9. CHECKLISTS

This chapter will give you step-by-step instructions to set up a simple multiplayer session from scratch.

Single Client Test Session:

1. Make sure to install the OMP package properly. Use vanilla Orbiter installs and check for .NET 2.0 framework installed.
2. Make sure your Windows OS supports TCP/IP connections.
3. Make sure your Internet connection is a direct one WITHOUT a HTTP proxy. Proxy support is not included right now.
4. Start Windows Explorer and go to your Orbiter base folder. Go to the /Server/Windows/ directory and double-click the “**ServerConsole.exe**” file. Your server should start now, showing you a simple console window with content like seen in this document. The server immediately starts to synchronize the internal virtual clock, so on machines with activated firewall, you may get a firewall message, if the server application should be allowed to connect to the internet. Click “Yes”, if you want the server to work, otherwise the application can not connect to a time server in order to synchronize to UTC.
5. Start Orbiter, go to the *Modules* tab and activate the **OMPClient** plugin. If you do not know how to do that, consider taking a look into Orbiter’s really fine manual or look at various tutorials found on the Orbiter forums. You should now see the client window. The client is starting synchronization immediately and therefore a possibly activated firewall will pop up a confirmation dialog. Again, click “Yes” to ensure synchronization to UTC.
6. Enter the IP address OR host name of your machine (that is: where the server resides) OR – in our simple test setup – 127.0.0.1 for the local host in the **Network tab page** of the client window. Click the **Connect** button in Orbiter’s main window.
7. A dialog will show up in the text window area, offering you an edit field and an empty list. You have to enter a name in the command line control for the vessel and to press ENTER within the command line control. If you leave the name blank and just press ENTER, your vessel will get your user name.
8. The next dialog will show you some locations your vessel can start from. Choose one by double-clicking on it. Orbiter will now start up. As soon as the render window is open, your simulation starts to synchronize MJD with the server.
9. As soon as your client caught up to the server’s MJD (you can use ‘R’ and ‘T’ as usual to accelerate faster – the clock controller will gradually decrease the acceleration as needed), you are online. You can now track your own vessels and play around with the client’s or server’s functions.

Multi Client LAN Test Session:

1. You need at least 2 machines connected via LAN to be able to run this kind of session.
2. Go through all steps of the *Single Client Test Session* for the server hosting machine, but be sure to set the firewall to allow UDP port 3000-3499 as well as TCP port 1515 for incoming messages.
3. On the second machine, go through the steps there, too, but skip step 4. Also enter another user name in the **Network tab page**.
4. As soon as both clients are online (i.e., caught up to the server's MJD – takes up to one minute…), you will see the opposite client's vessels (if they are in the same area, of course).

Internet Server Startup:

1. You need to have a machine directly connected to the internet and allowed to establish TCP and UDP connections. A static IP address is not necessary, but recommended (so you can keep the server running as long as you wish).
2. Go through steps 1-3 of the *Single Client Test Session*, but before starting the server in step 4, be sure to enter the correct Internet IP addresses in the **server.xml** file (instead of "*localhost*"). Also, be sure to setup you firewall/router to enable UDP port 3000-3499 and TCP port 1515 for incoming messages. Most probably, you'll have to establish **port-forwarding** to your server machine in order to succeed.
3. Change the admin password in the server.xml, too (you don't want others to manipulate your server, don't you?). Be sure to save the server.xml and continue with step 4 of the *Single Client Test Session*. Of course you can skip the client startup altogether if you only want the server running, but often you want to join the users flying on your server, too. In the later case, continue with client startup, but set the connection options appropriately in the **Network tab page**. It is highly probable that you'll have to use the Server-LAN option in order to see other clients. If you're seen by others, but can't see anyone yourself, chances are you either forgot the Server-LAN option, your chosen custom port is not forwarded to your machine, or the address for the server is wrong.

Internet Client Startup:

1. You need to have a machine directly connected to the internet and allowed to establish TCP and UDP connections.
2. Go through the steps 1-3 and 5 of the *Single Client Test Session*. Be sure to enter the correct IP address in **Server IP** and **Server Port** number in the **Network tab page**. Click the **STUN analysis** button and wait for the result to be shown in the text window. Follow the suggestions there to set your connection options properly. Then continue with the steps.
3. As soon as your client is online, you will see the other vessels, if in visible range. If not, you can use the **list** command to see who is online and what vessels are connected. To use it, enter "\list" (note the backslash) in the edit field of the OMP window and press ENTER.

10. FAQ

This chapter gives answers to frequently asked questions.

Q: I've started in fullscreen mode and switched to the desktop (to do something). After switching back, I get a white screen with only the OMPClient dialog showing up. How can I get my cockpit back?
A: Just close the OMPClient dialog by clicking the „X“ button to the right of the CL. You can get it back by pressing Ctrl-F4...

Q: I've joined a simulation by choosing a „Landed“ location and jumped right to a orbiting craft. But it seems like my velocity is not adjusted appropriately. What to do now?
A: This is due to the fact, that the Orbiter core cancels every velocity relative to the planet's surface while the vessel is still in „Landed“ mode. Despite the jump, the vessel remains in „Landed“ mode until you apply a small force, e.g. hover thrust. You can either apply the small hover BEFORE the jump, or apply the hover AFTER the jump and jump again.

Q: Where's the landing gear on my buddy's vessel?
A: Animations are not supported yet.

Q: My buddy sees a third vessel in a different position. What's wrong?
A: Vessels in orbital clusters must agree to a master and set this by the means of the „%master <ID>" command. Vessels at ground may suffer from a poor UTC and/or MJD synchronization. In this case, just wait for the clients to converge (can take up to 5 minutes with default settings).

Q: How do I run a local server without internet access for NTP synchronization?
A: You can specify only one SNTP server. Every client in the LAN session must specify this SNTP server, too, so everyone is running with the same – probably wrong – "UTC". It's the common time base that matters, not the name of the time base. If you use a W2K machine as server, you can easily use the built-in functionality – you have to activate it first, though (see for more information: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnw2kmag01/html/TimeWin2K.asp). If you use an XP machine, usually the built-in SNTP server is already activated. So, enter the machine's address in the local clients at the single **Server** element within the **NTP** element in the **OMPServerConfiguration** root of the client configuration XML, e.g.:

```
...
<Server Address="192.168.0.1" />
...
```

In the server configuration, just delete all Server elements. This will disable the server's NTP algorithm, as the local time is "UTC", anyway. If you want to set up an internet server, best practice currently is to define only 1 NTP server: the official NTP pool server at pool.ntp.org .

Q: Can I run the server on Linux in the background?
A: Yes, it is possible with e.g. "mono ServerConsole.exe console.xml > /dev/null &" . The missing stdin will suspend the input loop, but the operational threads will still run. Additionally, you can modify the log4net configuration to not use ConsoleAppenders, but only e.g. FileAppenders or even TelnetAppender to send the logs to different sinks then the console.

Q: I can't get beyond 0.1x time acceleration, although I'm connected to the server.
A: You are not fully connected to the server. This is a symptom of missing incoming UDP packets due to firewall/router/ISP blocking. You can check this by means of examining the "\list" response. Most probably you'll see your name there with an entry like "no UDP!". This shows that your UDP packets got through TO the server, but packets FROM the server to you don't. Although the hole-punching method should deal with these troubles, it is no silver-bullet and can fail, especially with cheap and/or old network-hardware along your route. Of course, very restrictive setups can have that effect, too. For this, the STUN analysis got implemented. Use it to determine your connection type and follow the suggestions.

Q: How can I port-forward my UDP ports through my router?
A: Answering this question for every possible network/vendor/hardware/version-combination you might have would fill books. Therefore, it is only possible to give hints like these:

1. Check your router's manual.
2. Most routers have web interfaces running on the routers IP.
3. Search for keywords like "services", "servers", "port-forwarding", "NAT", "routing", etc.
4. Every port-forwarding entry would consist at least of the port's number and the IP of the machine to route it to. That would be the number you've entered in the custom port edit field in the network tab of the OMP window and the LOCAL IP of your client machine. The former defaults to 2500, the later is probably along the lines of "192.168.x.y".
5. If there are additional settings, it mostly concerns the port protocol (UDP) and the destination port (same as the source port, i.e. the custom port from the edit field). Sometimes you can even specify ranges (just use the one port for an OMP client) and allowed source IPs (specify the equivalent of ALL then, maybe some wildcard like "*").

Q: Where is the official server?
A: There is none. At the time of the writing, though, there is a public server hosted by Ren Dhark on omp.dyndns-server.com, port 1515. You can also point your browser there, it has a nice web-page for the running ORL (Orbiter Racing League) hosting.

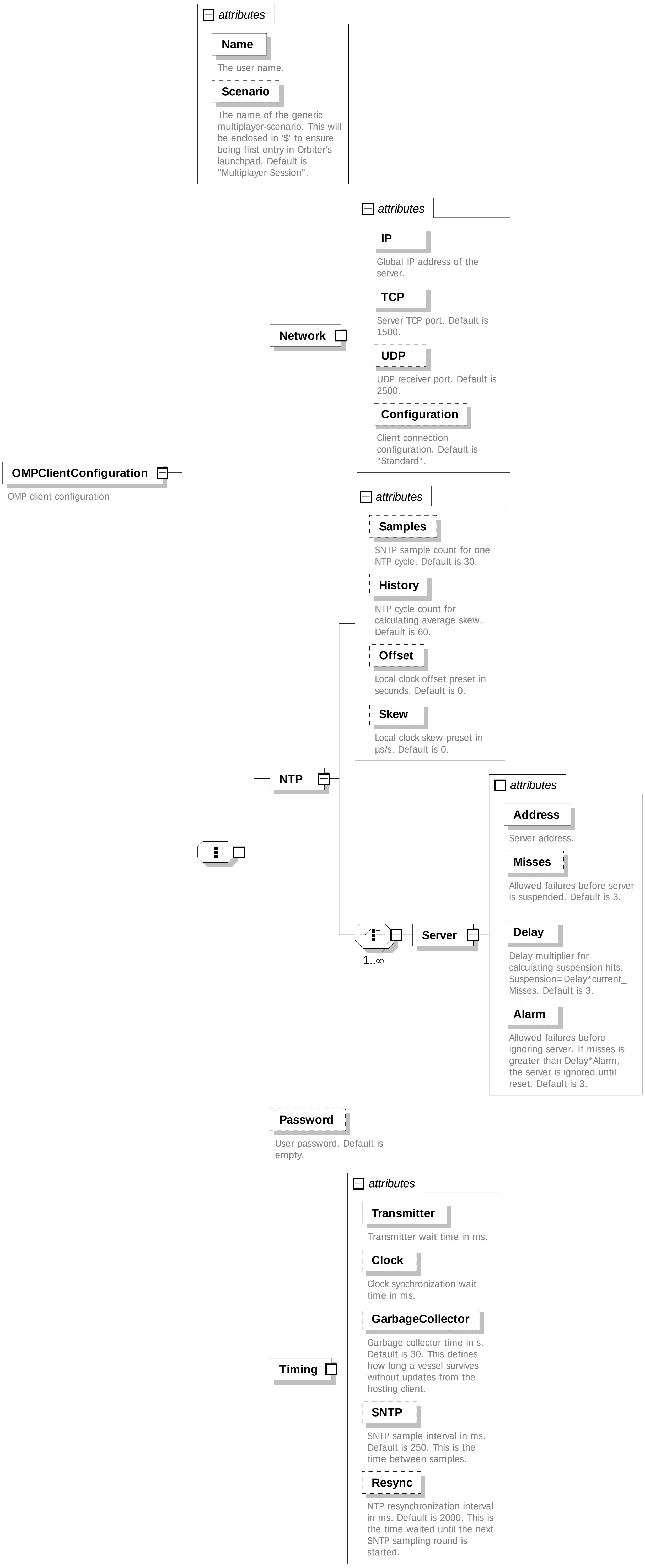
11. Appendix A

The following page depicts the syntax of the server’s configuration XML file.

attributes
Name
The name of the server.
Id
The servers ID.
OrbiterRoot
Path of the Orbiter installation root directory. Default is current directory.
SubvisualDistance
Subvisual distance - Maximum visibility distance in meters per size in meter. Default is 1000.
VerticalPixels
Amount of vertical pixels for calculation of subvisual distance. No default. Only if this together with FieldOfView is specified, it is taken into account, otherwise SubvisualDistance parameter is used.
FieldOfView
Angle between upper and lower ray from viewpoint through monitor edges. No default. Only if this together with VerticalPixels is specified, it is taken into account, otherwise SubvisualDistance parameter is used.
PromptFormat
Format of the prompt for all server wide messages. Default is "{0}", thus showing the raw message without timestamp.
OMPServerConfiguration
OMP server configuration
attributes
IP
Global IP address of the server. Needed for Server-LAN clients. Default is empty, causing system to use local end point IP.
TCP
TCP listener port. Default is 1500.
UDP
UDP receiver base port. Default is 2000.
Count
UDP receiver count. I.e. max concurrent client count.
Backlog
TCP backlog. Default is 10.
Web
Web server port. If not specified, no web server is started.
Network
attributes
Samples
SNTP sample count for one NTP cycle. Default is 30.
History
NTP cycle count for calculating average skew. Default is 60.
Offset
Local clock offset preset in seconds. Default is 0.
Skew
Local clock skew preset in µs/s. Default is 0.
NTP
attributes
Address
Server address.
Misses
Allowed failures before server is suspended. Default is 3.
Delay
Delay multiplier for calculating suspension hits. Suspension=Delay*current_Misses. Default is 3.
Alarm
Allowed failures before ignoring server. If misses is greater than Delay*Alarm, the server is ignored until reset. Default is 3.
Server
1..∞
attributes
Admin
Admin password for shell extensions and shutdown.
Container
Password for container clients.
Password
Messages
Welcome
Message on client connect.
Line
1..∞
Farwell
Message on client disconnect.
Line
1..∞
Shutdown
Message on server shutdown.
Line
1..∞
attributes
Transmitter
Transmitter wait time in ms.
GarbageCollector
Garbage collector wait time in s. Default is 30. This defines how long a vessel survives without updates from the hosting client.
SNTP
SNTP sample interval in ms. Default is 250. This is the time between samples.
Resync
NTP resynchronization interval in ms. Default is 2000. This is the time waited until the next SNTP sampling round is started.
Timing
attributes
Name
Name of the supported system.
Support
System
1..∞
attributes
Name
Name of supported vessel class. Special value '*' addresses all unconfigured vessel classes.
Redirect
Vessel class redirection target. Special value '#' and everything beginning with'@' mark ignored classes. The later is containing a name pattern for vessels to be deleted on client-side. The name pattern should contain a '*' as place-holder for the name of the vessel being deleted together with it. E.g. '@*_Bay' will delete XR2 bays.
Generic
Generic vessel class generator. E.g. Spacecraft3
Class
1..∞
Location
Generic scenario definition. See documentation Appendix C for details.

12. Appendix B

The following page depicts the syntax of the client’s configuration XML file.



OMPClientConfiguration
OMP client configuration

Name
The user name.

Scenario
The name of the generic multiplayer-scenario. This will be enclosed in '$' to ensure being first entry in Orbiter's launchpad. Default is "Multiplayer Session".

Network

IP
Global IP address of the server.

TCP
Server TCP port. Default is 1500.

UDP
UDP receiver port. Default is 2500.

Configuration
Client connection configuration. Default is "Standard".

NTP

Samples
SNTP sample count for one NTP cycle. Default is 30.

History
NTP cycle count for calculating average skew. Default is 60.

Offset
Local clock offset preset in seconds. Default is 0.

Skew
Local clock skew preset in µs/s. Default is 0.

Server

Address
Server address.

Misses
Allowed failures before server is suspended. Default is 3.

Delay
Delay multiplier for calculating suspension hits. Suspension=Delay*current_Misses. Default is 3.

Alarm
Allowed failures before ignoring server. If misses is greater than Delay*Alarm, the server is ignored until reset. Default is 3.

Password
User password. Default is empty.

Timing

Transmitter
Transmitter wait time in ms.

Clock
Clock synchronization wait time in ms.

GarbageCollector
Garbage collector time in s. Default is 30. This defines how long a vessel survives without updates from the hosting client.

SNTP
SNTP sample interval in ms. Default is 250. This is the time between samples.

Resync
NTP resynchronization interval in ms. Default is 2000. This is the time waited until the next SNTP sampling round is started.

12. Appendix C

Server configuration “Location” definition

The string in this definition is essentially the content of an Orbiter scenario file minus the header sections BEGIN_DESC / END_DESC and BEGIN_ENVIRONMENT / END_ENVIRONMENT. You can use every element valid for Orbiter scenario files, including addon-specific sections. In order to allow different locations, the system allows for special tags within the scenario file to define what elements are used in what locations. These tags will be processed whenever a client requests the start-up scenario file from the server.

The following tags are supported:

${IF <id>}	This is a conditional tag. Everything between this tag and a subsequent ${IF <id2>} or a ${FI} tag will be ignored (i.e. NOT written to the final scenario file), if <id> is not the chosen location. After used for filtering, the server is not sending this tag to clients. <id> is an integer value greater or equal to zero.
${DESC}	This tag can occur anywhere inside a ${IF <x>} region multiple times, but only the first ${DESC} entry works. Everything between this tag and the next carriage return will be treated as description string for the location <x>. This string shows up as entry in the location menu dialog on session start-up. If a location does not have such a tag, it will not show up and can therefore not be selected. After used for filtering, the server is not sending this tag to clients.
${NAME}	This tag will be replaced with the vessel name given by the user on start-up. This tag is sent to the client unaltered.
${CLASS}	This tag will be replaced with the vessel class selected by the user on start-up. This tag is sent to the client unaltered.
${FI}	This is just a closing tag for ${IF <id>}, i.e., everything between this and a subsequent ${IF <id>} tag will be written to the scenario file independent of the chosen location. After used for filtering, the server is not sending this tag to clients.