

**SIXTH FRAMEWORK PROGRAMME  
PRIORITY IST-2002-2.3.1.8  
Networked Audiovisual Systems**

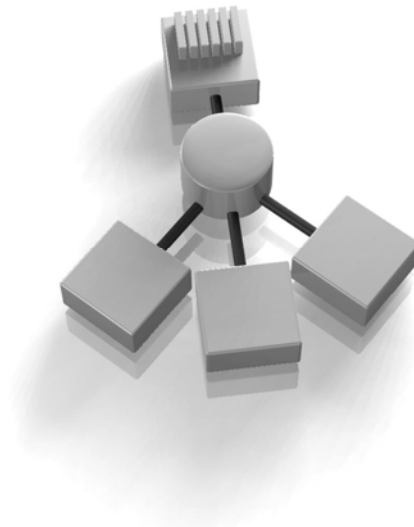


**Uni-verse project**

**Deliverable D 3.4**

**Tested implementation of Verse protocol**

**Feb 1, 2005**



STREP project

Project acronym: Uni-Verse

Project full title: A Distributed Interactive Audio-Visual Virtual Reality System

Proposal/Contract no.: 002228

Distribution: Public

## ***Content***

<b>Content .....</b>	<b>2</b>
<b>Summary .....</b>	<b>4</b>
<b>About the Author .....</b>	<b>4</b>
<b>Test Coverage .....</b>	<b>4</b>
<b>Result Reporting.....</b>	<b>4</b>
<b>General Purpose/Node Commands.....</b>	<b>5</b>
<b>Node-Specific Commands.....</b>	<b>5</b>
Object Node Commands .....	5
Geometry Node Commands .....	6
Material Node Commands .....	7
Bitmap Node Commands .....	7
Text Node Commands.....	7
Curve Node Commands .....	8
Audio Node Commands .....	8
<b>Conclusions .....</b>	<b>8</b>
<b>Availability.....</b>	<b>8</b>

<b>WORKPACKAGE DESCRIPTION</b>		
Workpackage Title: <b>Protocol/API extension</b>		WP No: <b>3</b>
Starting date: month <b>.3</b>	Duration: <b>9</b> months	Total Effort in Man-months: <b>21</b>
Member involved	Task description / Contribution of Member	Effort man-months
<b>KTH</b>	<b>Workpackage leader, Participant</b>	<b>12</b>
<b>II</b>	Participant	<b>1</b>
<b>HUT</b>	Participant	<b>1</b>
<b>FHG/IGD</b>	Participant	<b>1</b>
<b>MINUSPLUS</b>	Participant	<b>0</b>
<b>PAREGOS</b>	Participant	<b>0</b>
<b>BLENDER</b>	Participant	<b>6</b>

### Objectives

The current version of the real-time protocol developed by partner II [17] is event-based for graphics and animation. This WP will extend the protocol enabling high quality sound and acoustics modelling, animation and skeleton representation. To facilitate the integration of audio and acoustic simulation in the Uni-Vers system the protocol will be extended to enable the description of virtual sound sources, their behaviour and the acoustic properties of the modelled environment that interacts with the sound sources. Added to existing support for security encrypted login will be implemented. Evaluation and testing of protocol extension will be undertaken according to current software design practices. Delivered as Open Source. In this work package will also deliver the API for the scripting environment. Since the scripting environment is plug-in based this is important for other workpackages that will use it by writing plug-ins.

### Description of work / tasks

The work will comprise the following tasks:

#### WP 3.1 Protocol improvement

Initially a transform and an animation system for efficient low bandwidth communication of 3D motion data will be implemented. Adding audio support.

#### WP 3.2 Security

Implementation of an authentication mechanism.

#### WP 3.3 Scripting environment API

Implementation of the Basic API for writing processing plug-ins for the scripting environment

#### WP3.4 Testing

Testing of the protocol extensions and their implementation. Corrections, bug fixes etc.

### Deliverables

The deliverables of WP 3 are:

**D 3.1.** Implementation of the improved protocol (Month 5)

**D 3.2** Security implementation (Month 8)

**D 3.3** Implementation of the scripting API (Month 8)

**D 3.4** Tested implementation of protocol (Month 12)

### Milestone

**M 3.1** Tested implementation of protocol; Works according to D 2.2 (Month 12)

### Interrelation with other workpackages

Basic task for all subsequent workpackages. Provides input to WP4 -WP7.

# Deliverable D3.4 Tested implementation of Verse protocol

## **Summary**

This document, D3.4, is the description of the tests performed on the current implementation of the Verse protocol in Uni-Verse project, A Distributed Interactive Audio-Visual Virtual Reality System, proposal/Contract no.: 002228. What have been tested are the concrete implementations, in the form of the Verse standard API (for sending and receiving commands over a network connection) and the reference server (for serving as a back-end and data storage service). The below results refer to these pieces of software, often in conjunction with applications that exercise the relevant commands.

## **About the Author**

This report was written by Emil Brink, who is employed by partner KTH as a developer/engineer. Emil has a Master of Science in Computer Science and Engineering from KTH, and also worked with Eskil Steenberg on the original Verse implementation at the Interactive Institute in 1999 and onwards.

## **Test Coverage**

Testing of the Verse protocol is most easily described by listing the various parts of the protocol together with the testing status and results for that part. The protocol can be naturally divided into a set of general-purpose commands, plus a set of node-specific commands.

## **Result Reporting**

In the tables that make up the core of this document, individual Verse commands are listed by name, with a test status given for each tested command. Tested commands have been found to work, but that cannot be taken as conclusive evidence that they do indeed work; there could be hidden problems still. Tests have in most cases not been made directly, but rather by observation of other Verse clients. The development of these programs has several times uncovered errors in the protocol and/or reference host code, which have since been corrected. “Connector” is the name of an interactive Verse “data-browser” application, written by Eskil Steenberg, KTH. It is useful for many of these tests since it makes much node data, such as materials and curves, directly editable using a mouse-driven user interface. Since all its editing is done in a proper Verse fashion, by sending commands to a host and waiting for responses, if editing curves works (and it does), then the underlying commands can be assumed to work.

“Purple” is the code name for the D3.3 Scripting API implementation, written by Emil Brink, KTH. It is a fairly complex Verse client that downloads almost all data and makes it available to small plug-in programs that it then runs. Purple uses some Verse facilities, such as tags, methods, and text nodes, for its own interface needs, and allows almost all other things to be used by plug-ins and tested that way. This is the difference between “used by Purple” vs. “tested through Purple” that appears below.

There exists a plug-in (written by a member of the Blender open source community, Brecht Van Lommel) for the free image manipulation program “The GIMP” that connects it to Verse; it can be used to edit a bitmap stored on a Verse server remotely. It has been used to test the bitmap commands.

Many commands not often exercised by ordinary clients have been tested by the writing of dedicated (“special-purpose”) testing clients. This has often been done in the high-level programming language Python, using the Verse API bindings developed by B. Van Lommel.

## General Purpose/Node Commands

These are commands that act on a system level, or that are applicable to nodes of all types.

<i>Command Name</i>	<i>Status</i>
connect	Tested by all clients, part of connect.
connect_accept	Tested by all clients, part of connect.
connect_terminate	Tested when connection times out.
ping	Tested by special-purpose client.
node_list	Tested by all database-downloading clients.
node_create	Tested by editing clients, Purple, etc.
node_destroy	Tested in Connector.
node_subscribe	Tested by all database-downloading clients.
node_unsubscribe	Tested by special-purpose client.
tag_group_create	Tested in Connector, used by Purple.
tag_group_destroy	Tested in Connector.
tag_group_subscribe	Tested in Connector, used by Purple.
tag_group_unsubscribe	Tested by special-purpose client.
tag_create	Tested in Connector, used by Purple.
tag_destroy	Tested in Connector.
name_set	Tested by special-purpose client, often used.

Many of these commands are tested by every single Verse application, so they are perhaps the most well-tested of all commands. On the other hand, some of them are obscure and have received very little testing.

## Node-Specific Commands

The Verse data model defines seven different types of nodes that hold data; object, geometry, material, bitmap, text, curve and audio. This section lists the commands for each node type.

### Object Node Commands

These are the commands defined for working with object nodes. There are several categories of commands; transform (position, rotation, and scale), light, linking, methods, and animation control.

<i>Command Name</i>	<i>Status</i>
o_transform_pos_real32	Tested through Connector.
o_transform_rot_real32	Tested through Connector.
o_transform_scale_real32	Tested through Connector.
o_transform_pos_real64	Tested through Connector.
o_transform_rot_real64	Tested through Connector.
o_transform_scale_real64	Tested through Connector.
o_transform_subscribe	Tested through Connector.

<i>Command Name</i>	<i>Status</i>
o_transform_unsubscribe	Tested by special-purpose client.
o_light_set	Tested through Purple.
o_link_set	Tested in Connector, used by Purple.
o_link_destroy	Tested in Connector.
o_method_group_create	Tested in Connector, used by Purple.
o_method_group_destroy	Tested in Connector.
o_method_group_subscribe	Tested in Connector, used by Purple.
o_method_group_unsubscribe	Tested by special-purpose client.
o_method_create	Tested in Connector, used by Purple.
o_method_destroy	Tested in Connector.
o_method_call	Used (heavily) by Purple.
o_anim_run	Tested by special-purpose client.

Transform commands have only been tested interactively in Connector, which does not use the time stamping features. Method commands have been well-tested through the development of Purple, which uses object methods as its core controlling mechanism.

### Geometry Node Commands

These are the commands defined for working with geometry nodes. There are four categories of commands here: layer management, layer data setting, crease setting, and bone management.

<i>Command Name</i>	<i>Status</i>
g_layer_create	Tested by many graphics clients (incl. Purple).
g_layer_destroy	Tested in Connector.
g_layer_subscribe	Tested by many graphics clients (incl. Purple).
g_layer_unsubscribe	Tested by special-purpose client.
g_vertex_set_real32_xyz	Tested by special-purpose client.
g_vertex_delete_real32	Tested by special-purpose client.
g_vertex_set_real64_xyz	Tested by modelling client, and through Purple.
g_vertex_delete_real64	Tested by modelling client, and through Purple.
g_vertex_set_uint32	Tested by special-purpose client.
g_vertex_set_real32	Tested by special-purpose client.
g_vertex_set_real64	Tested by special-purpose client.
g_polygon_set_corner_uint32	Tested by many graphics clients (incl. Purple).
g_polygon_delete	Tested by modelling client, and through Purple.
g_polygon_set_corner_real32	Tested by modelling client, 3ds-loader, etc.
g_polygon_set_corner_real64	Tested by modelling client, 3ds-loader, etc.
g_polygon_set_face_uint8	Tested by special-purpose client.
g_polygon_set_face_uint32	Tested by special-purpose client.

<i>Command Name</i>	<i>Status</i>
g_polygon_set_face_real32	Tested by special-purpose client.
g_polygon_set_face_real64	Tested by special-purpose client.
g_crease_set_vertex	Tested through Purple.
g_crease_set_edge	Tested through Purple.
g_bone_create	Tested in Connector.
g_bone_destroy	Tested in Connector.

### Material Node Commands

These are the commands defined for working with material nodes. There are very few such commands, material definitions are rather “passive” with all the complexity contained in the data structures used to define fragments, rather than having a large set of commands.

<i>Command Name</i>	<i>Status</i>
m_fragment_create	Tested in Connector.
m_fragment_destroy	Tested in Connector.

### Bitmap Node Commands

These are the commands defined for working with bitmap nodes.

<i>Command Name</i>	<i>Status</i>
b_dimensions_set	Tested by GIMP plug-in, and through Purple.
b_layer_create	Tested by GIMP plug-in, and through Purple.
b_layer_destroy	Tested by GIMP plug-in, and through Purple.
b_layer_subscribe	Tested by GIMP plug-in, and through Purple.
b_layer_unsubscribe	Tested by GIMP plug-in.
b_tile_set	Tested by GIMP plug-in, and through Purple.

Bitmap commands have received quite a lot of testing both through the development of Purple, and also by the community-developed GIMP plug-in that allows real-time distributed painting.

### Text Node Commands

These are the commands defined for working with text nodes.

<i>Command Name</i>	<i>Status</i>
t_set_language	Used by Purple.
t_buffer_create	Used by Purple.
t_buffer_destroy	Used by Purple.
t_buffer_subscribe	Used by Purple.
t_buffer_unsubscribe	Used by Purple.
t_text_set	Used by Purple.

Most of the testing of text nodes comes as a side effect of the Purple project, which relies heavily on text nodes for parts of its interface needs.

## Curve Node Commands

These are the commands defined for working with curve nodes. It is important to realize that these commands deal only with the definition of curves, not with the actual usage of curves to animate things.

<i>Command Name</i>	<i>Status</i>
c_curve_create	Tested in Connector and through Purple.
c_curve_destroy	Tested in Connector.
c_curve_subscribe	Tested in Connector.
c_curve_unsubscribe	Tested by special-purpose client.
c_key_set	Tested in Connector and through Purple.
c_key_destroy	Tested in Connector.

Curve node testing has mainly been done interactively through the Connector application.

## Audio Node Commands

These are the commands defined for working with audio nodes.

<i>Command Name</i>	<i>Status</i>
a_layer_create	Tested by special-purpose client.
a_layer_destroy	Tested by special-purpose client.
a_layer_subscribe	Tested by special-purpose client.
a_layer_unsubscribe	Tested by special-purpose client.
a_block_set	Tested by special-purpose client.
a_block_clear	Tested by special-purpose client.
a_stream_create	Tested by special-purpose clients.
a_stream_destroy	Tested by special-purpose clients.
a_stream_subscribe	Tested by special-purpose clients.
a_stream_unsubscribe	Tested by special-purpose clients.
a_stream	Tested by special-purpose clients.

Audio testing has been done by special-purpose, rather simple, clients only.

## Conclusions

The Verse protocol has received a lot of testing during the past year, and is shaping up rather well. Core areas of functionality are working well, but there are still areas that have received less testing than would be desired; hopefully this will be addressed by project partners starting to use the new functionality.

## Availability

The special-purpose test clients described above are available in the public CVS:  
<<http://projects.blender.org/viewcvs/viewcvs.cgi/verse-tests/?cvsroot=verse>>.