3D-CONSOLE, AN APPLICATION INTERFACE BASED ON X3D

—

SUPPLEMENTS

Brian Kristiansen
Brian.Kristiansen@oracle.com

Kristian Spangsege
Kristian.Spangsege@framfab.dk

EP-THESIS-2003-38

IMM

Printed by IMM, DTU

Appendix A

Project description

English title:	3D-Console, an Application Interface based on X3D
Advisor:	Niels-Jørgen Christensen
Period:	1/9-2002 - 1/8-2003
Credit:	50 ECTS points
Participants:	Brian Kristiansen & Kristian Spangsege

English project description

We wish to implement a three-dimensional (3D) user interface to the basic tasks carried out on the operating system level. This 3D interface should provide spatial navigation in a simple virtual environment optimized for future HID's and not only for conventional mouse and keyboard.

This environment should feature the following:

1. Launching of applications.
2. Manifestation of applications in the virtual environment. Applications should represent themselves in the virtual environment, by showing a 3D model as entrance to the application, as well as a dynamically represent themselves while the application is running.
3. Have an API, which enables applications to interact with the environment.

Besides implementing the above features we want to discuss how this environment and navigating in it, could become comfortable, immersive and still be efficient for the advanced user. Efficiency and fitness for the experienced and technical user is the main goal. Our goal is not, like many others, to create an intuitive and recognizable virtual world, which is easy to understand for the non-technical user.

We will discuss how to enhance efficiency, comfort and immersiveness, by using different HID's and computer graphical effects.

It is our belief that working in a 3D virtual environment has the potential of enhancing the human to computer interaction.

Appendix B

ToDo

We should use gluTesselation to solve the non-convex faces of an IndexedFaceSet. gluTesselation might also be helpful in other situations. The Official Guide to Learning OpenGL, Version 1.1 - # Chapter 11. Tessellators and Quadrics (http://www.dcc.unicamp.br/~lmarcos/courses/mc603/redbook)

ExecutionContext::rootGroup and SubContextNode::subContext needs timestamps exactly like normal X3D fields. This is needed by Node::changedSince.

In the SAI we need to work with context type IDs exactly like for Nodes where the Node type ID is embedded within the Node ID. This is a requirement if a SAI application should be able to do the following:

```
Ref<ExecutionContext> context = ...;
Ref<Aplication> application = dynamic_cast<Application *>(context.get());
```

New idea for Ref (smart pointer). There are currently quite a few problems with the smart pointer due to the fact the it starts with a reference count of 0 and not 1. One problem is that it is impossible to distinguish between a newly created node and a node that has reached the point of destruction. This has become an issue with the increased use of multi threading. Another problem is that the constructor cannot create references to the object being constructed, since that may lead to the object being destroyed before the constructor returns, this is especially unfortunate for the sake of starting threads from a constructor with a BackRef to the object. A better solution may be to start the ref count on 1 and then require all new-calls to remove the extra ref count after assigning the pointer to a Ref. This would solve the mentioned problems (NO IT WOULDN'T - YOU MAY NEVER CREATE REF IN CONSTRUCTOR TO OBJECT BEING CONSTRUCTED) and also provide for the constructor Ref(T *) or the 'inc' method to check and barf if the reference count has dropped to zero. The problem is that it is extremely hard to find all the places there a reference counted object is newed in the existing code.

New idea for Thread object. Thread should be a reference counted object. You can then either run a function which will yield a Ref<Thread> or you can derive the Thread class and then instantiate it, then call run on it. In both cases the implicit or explicit run should increase the reference count of the Thread object and when the thread stops executing the reference count should be decreased. This will ensure that the Thread object lives as long as the thread is running. Also, it will free the instantiator of having to deal explicitly with the destruction of the Thread object. A good example is the SAI::ApplicationReaper. On the other hand, there will be situations where a BackRefObject 'O' is referring to a Thread 'T' which itself refers back to A with a BackRef. If T has a nature that makes it run forever if not explicitly terminated, then it will keep A alive for too long. This should then be handled by A::refForwardDestroy with a call to T::terminate(false). One example of this is the ExternalEventQueue of the X3D::SAI::Session. Another example may be the X3D::Loader. (THIS IS NOW IMPLEMENTED)

Alas, Nodes should not be associated with contexts. Nodes should be able to have different names in different contexts simultaneously! The current restriction for routes not to be between contexts thus cannot be maintained - what does the spec say about this? Probably the current context association of a node should be changed to a server association. But when a node does not have a unique association to a context, then how are relative URI fields resolved (the base URI is a property of the context), and why are nodes created by a context service, and why are routes created by a context service, when the source and target nodes need not have any other association with that context. Also, should the naming of a node induce a reference count on the node? Is it so that a node does have a unique association with a particular ExecutionContext, but may be freely entered into others? In this case, what happens when the context of the node gets unreferenced, should it remove its nodes from foreign contexts, or should the context stay alive as long as its nodes are in use?

Alas, Node identifies as seen by a SAI application must according to the X3D specification be unique across several applications connected to the same server. ISO_IEC_19775 Part 2, 4.4.8 "Node" says "Each node is defined by a unique identifier. This identifier is unique for that session. That is, it is possible that a single browser may be servicing multiple applications simultaneously and therefore all node identifiers are unique and invariant for the life of the session. This allows two external applications to potentially share data between themselves unambiguously and still have either make service requests of the browser with that shared data". Here a Session obviously refers to the life of the server and not the life of an application connection. This probably stems from the idea that it is the application that starts the server, and not vice versa.

There may be glitches somewhere in the code relating to the use of FieldBase pointers. The problem is that FieldBase pointers could point to CustomFields which may get deallocated resulting in a dangling pointer. In most cases this cannot happen since we hold a Ref to the node instance -- but are there cases where we do not? Probably a whole new strategy is required to forcefully prevent this.

Prevent cyclic node containment. This is not possible when parsing a scene, but cyclic containment may be introduced through routes and scripts and external applications.

It would probably be a good idea to add a glGetError() after each loop and write it out if non-zero.

Everywhere we use plain "int" to hold values of SFInt32. this is not acceptable for all platforms. We need a general solution to this problem.

Detect invalid names of custom script fields such as "initialize" and "shutdown".

Just some thoughts: Make a Server class for holding all the scenes of the "Browser". Put the "master mutex" in this server object also. This will make it accessible to the Script::onLoaded method and will enable it to initialize the SpiderMonkey engine asynchronously. Note: we could even postpone acquiring the "master lock"/"event cascade lock" until sending the actual events that may result from calling the initialize() script function. We would need an engine lock then. Also we should put a time controller in the Server object that ensures an ever increasing time stamp. Note the beauty of having the "always secure" up wards path "engine->node->context->server" (this is a general design trick). Note also how the beast of multi threading is tamed by the use of reference counting in the interaction between the UrlObjects and the loader jobs (See documentation for Utilities::Ref<T>::getSafeRef)

What kind of unicode encoding does the SpiderMonkey ECMAScript engine use: UCS2 / UTF16? (http://www.terena.nl/library/multiling/unicode/utf16.html)

Prevent cyclic containment within the scene graph. This is required by the X3D specification (October 22, 2002 - Part 1 - 4.3.5 Transformation hierarchy). It also prevents an otherwise subtle kind of memory leak relating to the use of smart-pointers. That is, if object A has a smart pointer referring to B, and B has one referring to A, then the two will live for ever, even when every one else has forgotten about their existence. This is because they both have a usage count greater that zero. They stay alive for each other but for no one else. This is an inherent problem with smart pointers. In fact it touches upon an existentialistic question: What is the real application and what is garbage? You could also say that, if you only have local knowledge about your relationship to others, then you won't know which party you belong to, the "real" one or to the garbage. The point is that to the object a relation ship to a garbage object seem as "real" as anything else. To deal with this problem of spawned entities (schizophrenia) we need a bigger picture and a more time consuming algorithm. Only when the scene graph is directed and acyclic, and there is a well defined origin node of the graph, then it will be possible to prevent memory leak through the use of smart pointers.

Phases in a nodes life. First one is Initialization phase. In this phase all fields regardless of access type can be assigned to, but assignments should not have any side effects. This concept is not supported yet. The obvious thing to do would be to add a flag to every node that indicated whether the node is in the initialization phase or it has been "realized". In the initialization phase it is important that those assignments that by definition calls functions in the node does not have any effect, This ensures that when parsing an X3D file then the assignment of values will not have any side effects. This includes calling script functions. Note also that this flag should disable event cascading, if it is possible via the SAI to assign to a filed after adding a route to that field while still keeping the node in the initialization phase. I'm unsure if one can add a route to a node that is in the initialization phase though.

XML parser/Script: Node values for custom fields as contained node elements. Support for deprecated USE attribute of field element.

Vertical text layout.

Support for concave polygons in Indexed face sets.

PixelTexture support.

The XML parser barfs about the USE attributes. Maybe they work anyway?

Don't inhibit color change and texturing just because a geometry node is unlit!

Switch to list available servers (SAI)

Switch to list available view point names

Switch to request screen dump from requested viewpoint and then terminate immediately (possibly with timed animation, i.e. after 2 seconds of animation).

Keystroke to make a screen dump.

Switch to set the path and stem of filenames of the saved screen dumps.

BUG: Normals may be wrong in anisotropically scaled coordinate systems. This seems to be revealed by the new "show normals" feature and is well illustrated by the "torus.x3d" scene. The big question is: Is it the show-normals feature that does not work, or is it the normals specified by glNormal that are wrong? I think it is the "show-normals" feature that is broken. Whether the normals specified by glNormal are automatically adjusted to compensate for anisotropically scaled coordinate systems can be examined by rendering a polygon in an unscaled coordinate system who's true normal is almost but not completely parallel to the up/down direction. This should result in a very dark and maybe even invisible polygon due to the small angle between the viewing direction and the polygon surface. Next step is to scale the local coordinate system by a huge factor in the up/down-direction. The larger the scaling factor the closer the true polygon normal gets to be parallel to the viewing direction. The result should therefore be a mush more light intensive polygon, and by increasing the scaling factor sufficiently much it should be possible to get arbitrarily close to the luminance produced by a polygon rendered normal to the viewing direction in an unscaled coordinate system. Also the clock test-application seems to reveal normal problems (sides of the seconds arrow are much too light intensive) is it the same problem?.

GIF image format lacks write support. GIF write support requires us to implement a color quantizer / half-toner (Floyd-Steinberg dithering)

Do buffering in the client application between beginUpdate and endUpdate instead of sending each change over the network individually. This should be part of the abstraction layer.

Use a scheme of auto-generation similar to that used for the client applications to generate a scene graph cache. This could be a basis for implementing multiple viewers.

UPS it seems like the omniORB if fully capable of spawning threads to serve incoming requests without any thread calling the "run" method on the ORB. Probably it spawns its own thread on systems with pthread support. The problem is that we have no control over when incoming requests can start to occur.

GL_DITHER is by default enabled. It must be disabled when we want to render for pointing device collision detection. Also one article mentioned that dithering should be disabled when clearing the "screen" to increase performance.

Named X3D nodes are never deallocated before the execution context is. Further more the name stays mapped in the context. This is all due to the fact that the mapping holds a reference to the node, and the ref-count then never goes to zero. Also the node destructor does not remove the name mapping.

Proper support for multiple strings in X3D::XML parser (WHAT DOES THE "X3D IN XML" SPEC SAY?):
<ImageTexture url='"../../vts.jpg"
"http://www.web3d.org/TaskGroups/x3d/translation/examples/Conformance/vts.jpg"'/>

The fact that we refer to SensorGroup with a plain pointer allows it to be deallocated between frames. This can be remedied by using a Ref.

Implement dynamic viewpoints. When viewer is bound to a moving viewpoint the viewer should move with it. Probably the viewer should be released from a viewpoint as soon as the viewer chooses to move (translational navigation - not rotation, field of view change).

Implement the effect of the radius of a point light (but see x3d_spec_peculiarities)

Client restrictions:
- Quota on number of polygons.
- Quota on number of nodes.
- Quota on number of scene graph depth.
- Some nodes require special permission to insert into the scene graph:
  Lights, background, viewpoint

A PermissionException should be added to most SAI operations.

There are some quality problems when using mipmaps together with automatic texture coordinate generation.

Setting the system clock stops the TimeSensor! It needs to be more robust.

We need to come up with a functionality to lay out applications in the VR environment. As we now are able to start applications from within the environment, we need at way to control where the applications are situated at startup. (One way could be to display an wire frame model of an bounding box representing the applications dimensions. This box could be more or less meshed. This box could then be dragged to the position/orientation that is desired.)

Appendix C

Dependencies

```
Except for absolute core requirements such as OpenGL and SDL we should
allow the features depending upon these libraries to be selectivly
included through build configuration. Proberbly this will go hand in
hand with the application of GNU Autoconf.


OpenGL or Mesa3D:
-----------------

http://www.opengl.org/
http://www.mesa3d.org/

A 3D graphical API, OpenGL is a SGI product and Mesa is an open source
initiative. Mesa is an integrated part of XFree86.

Tested with XFree86 4.2.0 (I do not know which version of Mesa this is,
except that it version 3.x.x)

Is used in "x3d/view.C", "x3d/geometry.C", "x3d/text.C" and
"x3d/aux.C" and a bit in "3dconsole/scene_view.C". Nowhere else.


SDL (Simple DirectMedia Layer):
-------------------------------

http://www.libsdl.org

A somewhat platform independant multimedia library, that includes
grpahical screen, input devises and sound APIs.

Tested with version 1.2.2.
Tested with version 1.2.3 which is default on Redhat 7.3.

It is used in "utilities/window.C" only.
```



```
EXPAT:
------

http://www.jclark.com/xml/expat.html

An XML parser.

Tested with  version 1.95.1 which is default in Redhat 7.2.
Tested with  version 1.95.2 which is default in Redhat 7.3.

Is used in "x3d/parse_xml.C" only.


LIBPNG:
-------

http://www.libpng.org/pub/png

A library for loading and saving PNG images.

Tested with version 1.0.12 which is default on Redhat 7.2 & 7.3.

It is used in "utilities/image_png.C" only.


LIBJPEG:
--------

http://www.ijg.org (Independent JPEG Group)

A library for loading and saving JPEG images.

Tested with version 6b which is default on Redhat 8.0.

It is used in "utilities/image_jpeg.C" only.


LIBUNGIF:
---------

http://prtr-13.ucsc.edu/~badger/software/libungif

A library for loading and saving GIF images.

Tested with version 4.1.0 which is default on Redhat 8.0.

It is used in "utilities/image_gif.C" only.


W3C-LIBWWW:
-----------

http://www.w3.org/Library

A library for down and uploading through the HTTP protocol and other
standard protocols among those named by the various possible URI
schemes.
```

```
Tested with version 5.3.2 which is default on Redhat 7.3.
Tested with version 5.4.0.

w3c-libwww-5.2.8 which is standard on ReadHat 7.2 has a bug that
hangs the X3D loader when loading multiple files from the local disk
(ie. the file URI scheme.) A work around is to use the command line
option -t 1.

This problem has disappeared in version 5.4.0, so if you experience this problem
upgrade to version 5.4.

It is used in "utilities/uri_request.C" only.


OmniORB4:
---------

http://omniorb.sourceforge.net
(http://www.corba.org)

Is a CORBA Orb.

Tested with a gcc3.2 build of version 4.0.0.

Is used throughout "3dconsole/*".


FreeType:
---------

http://freetype.sourceforge.net

Is a font rendering engine

Tested with version 2.1.2 (default on RedHat 8.0).

Is used in "x3d/text.C" only.


SpiderMonkey:
-------------

http://www.mozilla.org/js/spidermonkey
ftp://ftp.mozilla.org/pub/js (download)
http://www.mozilla.org/js (What is JavaScript/ECMAScript?)

Is an ECMAScript engine.

Is tested with version 1.5rc4.

On RedHat 8.0 you will need to install these two packages:
libjs-1.5rc4-3.i386.rpm
libjs-devel-1.5rc4-3.i386.rpm


Other system dependencies:
```



```
==========================

ICONV:
The "iconv" is utility for converting strings between character encodings.
It conforms to the UNIX98 standard.
It is used in "utilities/unicode.C".

POSIX THREADS:
The POSIX threads library (pthreads) is used in "utilities/thread.C".

TERM:
Terminal width determination in "utilities/term.C".

POSIX TIME:
POSIX timeofday -> "utilities/time.H", pselect -> "utilities/uri_request.C".

Various other UNIX related stuff
```

Appendix D

X3D mailing list discussions

```
X3D mailing list discussions:
======================================================================

(Note: The following mail has not yet been submittet to the mailing
list)

Hi, I have a couple of questions for the X3D Specification, ISO/IEC
19775:200x. Please note that I am primarily concerned about the
implementation of the specification and less about content writing.
Also note that most of my questions deal with external interaction
through the SAI.

Note that when I write ExecutionContext below I really mean any of the
following: The root scene of the browser, an instantiated prototype,
an inlined scene, the result of any of the createX3DFromXXXX services
of the SAI or the result of the createScene service of the SAI.

From the list of SAI services it is apparent that an ExecutionContext
deals with at least three things: Named nodes, routes and base
URIs. Can you agree with this?

My first question is concerned with the various aspects of the
relationship between Nodes and ExecutionContexts.

Of cause one such relation is the one that can be create by adding to
an ExecutionContext an association from a name to a Node. But let's
leave this for now.

My question realy is: Without regard to the name-relation mentioned
above, should we think of an ExecutionContext as a kind of Node
container. That is, is there any containment relation beween
ExecutionContexts and Nodes apart from the name-relations mentioned
above. If YES the what is the nature of this containment relation?
Which is true:
```



```
1   A node is a child/element of a particular and fixed
    ExecutionContext at all times during the lifetime of the node!

2   If not, then a node is a child/element of some (but not necessarily
    the same) ExecutionContext at all times during the lifetime of the
    node!

3   If not, then a Node is an entity that may act as a child/element of
    zero, one or more ExecutionContexts simultaniously, but has no unique
    ExecutioContext relation!

4   If not, when what?

It occurs to me that the specification is rather un-clear about
this. In fact I can find sections that (to me) supports any of the
alternatives above. I'll get back to this.


To me the nature of this containment relation (if any) is important
for two reasons: Because it impacts when and how a node can access an
ExecutionContext, and becase it impacts when and how a node reference
can be passed from one ExecutionContext to another.


ISO_IEC_19775 Part 2, 5.3.18 SAI_NODE_IN_USE says:

An alternative use of this error shall be to indicate that the node,
or one of its children are currently in use in another scene. It is an
error to share a single node instance across multiple scenes
simultaneously.



======================================================================


Hi,

I am currently in the process of implementing parts of the X3D
specification into a product called 3D-Console.

It seems to me that several aspects of the concept of named X3D nodes
is vaguely specified.

First of all I assume that an X3D node, created without an explicit
name, has no name. Is this true? Or, does it get an auto-generated
unique name? I can't seem to find any indication of this in the spec.

Next, I assume that node names must be unique within a specific
namespace. Is this true? The VRML97 specification seems to not require
this (VRML97 spec. 4.6.2 DEF/USE semantics).

If the last assumption holds true, then I see a problem in the case
where a named node is transferred by a route from one namespace to
another. What happens if the name clashes with another name in the
target namespace for the route? Does it invalidate the original
update? (proberbly not). Or does the event get silently discarded?
```

If nodes could have different names in different namespaces, that
would help solve the above problem, but other parts of the X3D
specification seems to rely on the fact that the name is a property of
the node and not a property of the binding of a node to a namespace.
(X3D spec. Part 2, A.5.3.1 getName()/getType())

Also, I have a number of smaller comments:

- SFChar and MFChar are listed as field types in the X3D specification
part 1.5, but why is SFChar and MFChar neither mentioned in the
specification of the VRML encoding nor in the specification of the XML
encoding
(http://www.web3d.org/TaskGroups/x3d/translation/x3d-compact.dtd)

- What is the difference between SFString and MFChar? Why are both
type needed?

- Why is MFImage not mentioned in the specification of the VRML
encoding of X3D?

- Why is the default value for SFTime -1 and not 0. To me 0 would be
the canonical choise, both with respect to the meaning of 0 as the
start of the UNIX epoc, and functionally.

- Why is SFUrl and MFUrl not mentioned in the field types in the X3D
specification part 1.5? And what is the default value for such fields?
And what is the meaning of the default value? CWD, none...

- Why is SFUrl or MFUrl not used in the Background node type?

Finally I would like to ask you where to direct questions about the
spec (as opposed to constictive feedback)?

Kristian Spangsege

Reponse:

Kristian Spangsege,

Thank you for all of your comments and questions. I will answer a few
of your questions. Others will respond to the remaining ones. The
spec is undergoing final editing prior to submission to the Web3D
Consortium members. The version that is posted is a bit out of date.
The new version will be posted next month. Your questions and
comments have been received early enough to have an impact on this
editing cycle.

Comment:

>Hi, I am currently in the process of implementing parts of the X3D
>specification into a product called 3D-Console. It seems to me that
>several aspects of the concept of named X3D nodes is vaguly specified.
>First of all I assume that an X3D node, created without an explicit name,
>has no name. Is this true?

If by "name" you are referring to the one assigned by DEF, then yes that is
true.

>Or, does it get an auto-generated unique name? I can't seem to find any
>indication of this in the spec.

There is nothing that prohibits (or requires) you to internally label nodes.

>Next, I assume that node names must be unique within a specific namespace.
>Is this true? The VRML97 specification seems to not require this (VRML97
>spec. 4.6.2 DEF/USE semantics).

You are correct about VRML97. I believe that the most recently defined
node was the one used. I don't recall (nor could I find) anything in the
current spec that requires a unique definition. One of the people more
familar with the spec should comment here.

>If the last assumption holds true, then I see a problem in the case where
>a named node is transferred by a route from one namespace to another. What
>happens if the name clashes with another name in the target namespace for
>the route? Does it invalidate the original update? (proberbly not). Or
>does the event get silently discarded?

Nodes do not get transferred by a route. The event generated (or consumed)
by a node does. There is no mechanism for transferring events from one
namespace to another, unless it is defined through the namespace's
interface. Note that a namespace is defined by the file the node resides
in. Each file is a separate namespace. Also, any PROTOs in a file are in
their own namespace.

>If nodes could have different names in different namespaces, that would
>help solve the above problem, but other parts of the X3D specification
>seems to rely on the fact that the name is a property of the node and not
>a property of the binding of a node to a namespace. (X3D spec. Part 2,
>A.5.3.1 getName()/getType()

A name is attached to a particular node within a namespace.

>Also, I have a number of smaller comments: - SFChar and MFChar are listed
>as field types in the X3D specification part 1.5, but why is SFChar and
>MFChar neither mentioned in the specification of the VRML encoding nor in
>the specification of the XML encoding
>(http://www.web3d.org/TaskGroups/x3d/translation/x3d-compact.dtd) - What
>is the difference between SFString and MFChar? Why are both type needed? -
>Why is MFImage not mentioned in the specification of the VRML encoding of
>X3D? - Why is the default value for SFTime -1 and not 0. To me 0 would be
>the canonical choise, both with respect to the meaning of 0 as the start
>of the UNIX epoc, and functionally. - Why is SFUrl and MFUrl not mentioned
>in the field types in the X3D specific!
>ation part 1.5? And what is the default value for such fields? And what is
>the meaning of the default value? CWD, none... - Why is SFUrl or MFUrl not
>used in the Background node type?

Tony, Alan, Joe???

>Finally I would like to ask you where to direct questions about the spec
>(as opposed to constictive feedback)? Kristian Spangsege

```
The x3d-contributors list is a very good place to start.  All of the spec
authors read that list plus some other people.  Subscriptions instructions
can  be obtained by sending an email to x3d-contributors-request@web3d.org
with the text "info" in the body.

+-------------
| Leonard Daly <daly@realism.com>
| Project Management http://realism.com/
| e3d News Technical Editor http://e3dNews.com/
| "Introducing X3D" SIGGRAPH Course Organizer & Presenter
+------------------------------


Elaboration of my question about the node names:
-------------------------------------------------
Thank you for responding so soon.

I hope it is allright that I reply directly to you, in stead of going throug
the "feedback system".

I have a few elaborations on my concern about the node names
(those assigned by DEF.)

You say that nodes do not get transferred by a routes.
But consider the following VRML snippet:

DEF group1 Group
{
  children []
}

DEF group2 Group
{
  children []
}

ROUTE group1.children TO group2.children

Now lets imagine that a script creates a new node with the name "foo"
and that it somehow gets added to "group1" via the "addChildren"
EventIn. Then, what I am talking about, is that the node named foo gets
added to "group2" due to the existance of the route.

The problem that I see arises when "group1" and "group2" lives in
different name spaces. That is, "group2" could have been some group
inside a PROTO instance. Of, cause it would not be as straight foreward
to attach the two by a route now, but as I understand the spec. it is
still possible using a route inside the PROTO.

Now, the reason that I am focussing specifically on the indirect
adittion by an event traveling through a route is that in this case
it is not as easy to abort the update as it is if the name clash is
detected at the original update operation (say through the
SAIs EventIn.setValue()).

Still, this whole issue is hardly an issue if names need not be unique
inside a specific namespace. But in taht case, many new questions
```



```
would arise for me.

So, do you catch my drift? I hope so. :-)


Kristian Spangsege


Regarding SFUrl and MFUrl:
----------------------------------------------
-----Original Message-----
From: Tony Parisi
To: Leonard Daly; x3d-specification@web3d.org
Cc: kristian.spangsege@framfab.dk
Sent: 9/29/2002 1:10 AM
Subject: RE: [x3d-specification] X3D Comment from Kristian Spangsege

> >of the UNIX epoc, and functionally. - Why is SFUrl and MFUrl not
> mentioned
> >in the field types in the X3D specific!
> >ation part 1.5? And what is the default value for such fields?
> And what is
> >the meaning of the default value? CWD, none... - Why is SFUrl or
> MFUrl not
> >used in the Background node type?
>
>
> Tony, Alan, Joe???
>

SFUrl and MFUrl are no longer in the X3D spec. You must be looking at a
version prior to the latest (7/21/02).

My reply:
---------
Hi, and thank you for responding.

Actually I am looking in the latest released specification.
"Milestone 4 - July 21, 2002" more specifically.

And indeed both SFUrl and MFUrl field types are mentioned in
numerous places in the specification. The following is a list
of some:

Part 1.17.3.2 ImageTexture
Part 1.17.3.3 MovieTexture
Part 4.5.1.1 Table of contents
Part 4.5.12 SFUrl and MFUrl
Part 4.A.4 Fields

Anyway, the important message from my point of view is that the
SFUrl and MFUrl fields aught to not be in the specification anymore.
Thanks again.


Kristian Spangsege

PS Maybe you could elaborate just a little on the reason for
```

abandoning the URI types.

==

Hi,

A have some questions about the concept of named X3D nodes (those
specified with DEF).

Note that in the following a refer exclusivly to the X3D
specification, Milestone 4 - July 21, 2002

Especially I am concerned about uniqueness of names within a
namespace. Therefore my first question will be:

What happens when several nodes are defined with the same name in a
VRML file like this:

```
DEF group Group {}
DEF group Group {}
```

As I understand it both group nodes live in the same namesapce. So
hwat happens:

- Should this yield a parse error?

- Do both nodes get the name "group"? In this case there must be some
  rules that determine which group node is aquired through the SAI
  operation Browser.getNode("group").

Whap happes if a scene defines a group node named "group" and a SAI
operation attepts to create a new node named "group" and inserts it
into the same namespace in the scene?

For now I assume that in all cases above the name clash is dealt with
by raising a parse error/exception. In that case I have another
question relating to the concept of passing node references along
routes. Considder the following scenario:

```
DEF group1 Group {}
DEF group2 Group {}
ROUTE group1.children TO group2.children
```

Now lets imagine that an ECMA script creates a new node with the name
"foo" and that it somehow gets added to "group1" via the "addChildren"
EventIn. Then, what I am talking about, is that the node named foo
gets added to "group2" due to the existance of the route.

The problem that I see arises when "group1" and "group2" lives in
different name spaces. That is, "group2" could have been some group
inside a PROTO instance. Of, cause it would not be as straight foreward
to attach the two by a route now, but as I understand the spec. it is
still possible using a route inside the PROTO.

Now, the reason to why I am focussing specifically on the indirect
adittion by an event traveling through a route is that in this case
it is not as easy to abort the update as it is if the name clash is
detected at the original update operation (say through the
SAIs EventIn.setValue()).

Still, this whole issue is hardly an issue if names need not be unique
inside a specific namespace. But in that case, many new questions
would arise.

==

To: x3d-specification@web3d.org, x3d-contributors@web3d.org

Submitter Information:

Name: Kristian Spangsege
Organization:
E-mail: kristian.spangsege@framfab.dk

X3D Feedback:

Comment:

Hi,

I have a number of comments and questions for the latest specification
(October 22, 2002). I hope you can use them as constructive feedback,
but I would also apreciate if someone would reply with answers.
Thanks in advance
- Kristian.

PS If there is a more (or just another) appropriate list for this kind
of feedback, would you plz. inform me?

- The lifecycle of a node.

  In the latest specification in Part 2 - 4.4.8 you state that the
  "realised" state indicates that "The node is participating in the
  scene graph and/or scripting". The you go on an state that:

  "At the point the user does anything with the node reference other
  than set the field values, the node shall transition to the realised
  state. For example, the user creates a Box node, sets the size
  field, creates a Shape node then immediately adds the Box to a Shape
  node, shall result in the state of the Box node changing to
  Realised, while leaving the Shape node in the setup state."

  I think there is some inconsistency in the wording here. Maybe it is
  just that your definition of "scene graph" is counter intuitive to
  me.

  I would naturally think of the scenegraph as all those nodes that
  are either root nodes of the scene or children of those root nodes

and the nodes currently referenced by scripts in the scene.

Consequently I would not think of the Box node in the "loose" shape node described above as being part of a scene graph. Yet you state that the "realized" state indicates that a node is part of the scene graph.

Which is true?

Also, what happens when the Shape node get removed from the scene graph? Does the transistion to "disposed" state imediately cascade to sub nodes like the Box node?

- A nodes association to a scene.

  In Part 2 - 5.3.18 you state that "It is an error to share a single node instance across multiple scenes simultaneously."

  This seems to suggest that a node has an explicit association to a specific scene. But does it imply that a node at all times is associated with a specific scene? Eg., is a freshly created node also associated with a scene? This seems to be suggested by the fact that (6.4.7 createNode) is an execution context service. What i'm after here is - does the specification have an implicit requirment that a node at all times is associated with some scene. The natural alternative would be that a node is associated with a scene when and only when it is part of the scene graph of that scene.

  Next, is it allowed to create a node through one execution context/scene and then insert it into the scene graph of another scene? Or more generally, can the association of a node to a scene be changed (eg. via the SAI)?

  Consider a route linking fields of type SFNode or MFNode, and whos source field and target field belong to nodes in two different scenes. Asuming that such a route can be validly created, that happens when nodes get "transported" along the route from one scene to another? The obvious result would of cause not be acceptable, since a node cannot be a part of two scenes simultaneously. But how is this situation to be handled?

- Restrictions in creation of new routes.

  The fact that dynamic route handling is an execution context service suggests that routes have a strict assiciation with a scene, and in turn this suggests that routes may not be added between two nodes in different scenes. Is this so?

  Disregarding SFNode and MFNode, is it then true that routes can only be created if the source field and the target field have exactly the same type?

  How are field value domain restrictions handled? Is it legal to create a route from an SFFloat field with no restrictions on the value domain to another SFFloat field with a restricted value domain (eg. Material.ambientIntensity)? If so, what happens during event

  cascade in case of a domain violation?

  In the case of SFNode or MFNode, is it true that an SFNode field cannot be routed to an MFNode field?

  Now, considder two in/out MFNode fields. Is it always allowed to create a route between them, or does it depend on the node type for the field (ie. X3DChildNode for Transform.children)?

  Asuming that the validity of the route depends on the node type of the fields, is the requirement that the node type of the two fields must be the same, or is it allowed as long as the source node type is a descendant (of order 0 or more) of the target node type?

  Lastly, if the node types for the fields are not checked during creation of routes, how are violations handled during event cascade?

- Node identifier uniqueness.

  In Part 2 - 4.4.8 you state that:

  "Each node is defined by a unique identifier. This identifier is unique for that session. That is, it is possible that a single browser may be servicing multiple applications simultaneously and therefore all node identifiers are unique and invariant for the life of the session. This allows two external applications to potentially share data between themselves unambiguously and still have either make service requests of the browser with that shared data."

  What exactly is meant by "session" here? From the definition of "Session" in Part 2 - 4.4.5 the two external applications represent two distinct sessions within the browser. So if two external applications must be able to exchange node identifiers the node identifiers mut be unique not just "for that session" but across all session within the browser, that is "browser wide".

  Ain't that true, or did I miss something?

- What does the "getName" node service return for unnamed nodes (Part 2 - 6.6.2)?

- Are node names case sensitive?

- Are field names case sensitive?

- Point light radius.

  How does the radius of a point light influence shading? On a per pixel basis? On a per object basis? And in the latter case how exactly should it be determined if the object is withing the radius of the light source?

Response:

>   I think there is some inconsistency in the wording here. Maybe it is

>   just that your definition of "scene graph" is counter intuitive to
>   me.

For the purposes of X3D, the scene graph is any collection of nodes put
together in a heirarchy.

>   Consequently I would not think of the Box node in the "loose" shape
>   node described above as being part of a scene graph. Yet you state
>   that the "realized" state indicates that a node is part of the scene
>   graph.

It is a difficult enough task keeping track of what nodes are available
where and when they are live or not. So, the simplest rule is to make
sure that when a node is added to any other node, it now assumes that it
is in a state where it could be run. If you do not do that, then every
node instance also needs to keep flags about describing whether it is
live or not.

>   Also, what happens when the Shape node get removed from the scene
>   graph? Does the transistion to "disposed" state imediately cascade
>   to sub nodes like the Box node?

Not necessarily. When the parent node moves to the disposed state, the
children also need to evaluate what state they are in. Some children may
still be part of the live scene graph due to the DEF/USE type of
construct. You can't just make everything disposed.

>   This seems to suggest that a node has an explicit association to a
>   specific scene. But does it imply that a node at all times is
>   associated with a specific scene?

It does not. It only states that a node instance can be in one scene at
a time. For example, if you use createX3DFromString you need to remove
the node from the returned scene and add it to the current scene. In
between it is not owned by any scene.

>   Next, is it allowed to create a node through one execution
>   context/scene and then insert it into the scene graph of another
>   scene? Or more generally, can the association of a node to a scene
>   be changed (eg. via the SAI)?

Yes. See above. If not, createX3DFromXXXX is useless.

>   Considder a route linking fields of type SFNode or MFNode, and whos
>   source field and target field belong to nodes in two different
>   scenes. Asuming that such a route can be validly created, that
>   happens when nodes get "transported" along the route from one scene
>   to another?

The addRoute call adds the route instance to this scene. Can't remember
the spec for this bit offhand, but I think it is specified to issue an
error if both nodes are not in the current scene. Many reasons for this
- including preventing someone from adding a route directly from the
internals of a PROTO to the outside scene graph.

> - Restrictions in creation of new routes.
>
>   The fact that dynamic route handling is an execution context service

>   suggests that routes have a strict assiciation with a scene, and in
>   turn this suggests that routes may not be added between two nodes in
>   different scenes. Is this so?

Yes. See above.

>
>   Disregarding SFNode and MFNode, is it then true that routes can only
>   be created if the source field and the target field have exactly the
>   same type?

Correct. This has always been a requirement - both VRML97 and X3D.

>   How are field value domain restrictions handled?

There are none. It is up to the content author to make sure they don't
do something stupid like that. If invalid values are passed along the
route, the browser is obligated to barf.

>   Lastly, if the node types for the fields are not checked during
>   creation of routes, how are violations handled during event cascade?

That's a browser implementation issue. The spec only states what the
valid field value are and then the browser decides how to issue the
error when it detects something wrong.

>   What exactly is meant by "session" here? From the definition of
>   "Session" in Part 2 - 4.4.5 the two external applications represent
>   two distinct sessions within the browser.

A session is an artificial construct in the spec to provide a way of
describing a connection between some form of user code and the browser
itself.

> So if two external
>   applications must be able to exchange node identifiers the node
>   identifiers mut be unique not just "for that session" but across all
>   session within the browser, that is "browser wide".

You cannot exchange nodes between two sessions. Creating a node in one
browser and shipping it across to another browser is an error.

>
> - What does the "getName" node service return for unnamed nodes (Part
>   2 - 6.6.2)?

If you don't have a name, you can't access the node. What argument would
you pass as the name for an unnamed node?

> - Are node names case sensitive?

Yes. See the core spec.

> - Are field names case sensitive?

Yes. See the core spec.

> - Point light radius.

```
>
>   How does the radius of a point light influence shading? On a per
>   pixel basis?

Lighting calculations are made on a per-pixel basis.

>   And in the latter case how
>   exactly should it be determined if the object is withing the radius
>   of the light source?

There is no way of doing this within X3D. You would need to write some
custom code to do so.

--
Justin Couch                         http://www.vlc.com.au/~justin/
Java Architect & Bit Twiddler              http://www.yumetech.com/
Author, Java 3D FAQ Maintainer                  http://www.j3d.org/
```

Appendix E

X3D specification obscurities

X3D specification - unsolved issues:
==

- Why does the X3D spec ISO_IEC_19775 Part 2, 6.5.5 "rootNodeHandling" say that "If the action is to remove and the node is not a known root node then this shall generate an error."? This is impossible since the request is buffered!

- Scripting. When a script handles an incoming event and generates out-bound events, the specifications say that the out-bound events are not delivered until the execution of the event handler terminates. Now, this means that the value must be buffered somewhere. The question is then, when the target field is read by the event handler, does it read the buffered value, or does it read the true value of the field. If reading gives the buffered value, we must be sure to fall through to the real value until the first buffered value is written. And what about if there are multiple executing threads in the same script? Do they all write to the same output buffer, or do each thread buffer outgoing events independently?

- What happens with a route whose target is a node that gets removed from scene graph? Does the fact the there still is a reference to the node (via the route) mean that the node must "live on", or should the routes be broken and the node destroyed? What about TimeSensors and Script nodes, may they keep sending events after being removed from the scene graph and until they are destroyed (say by a garbage collector), or should they stop immediately in sending events when they are removed from the scene graph?

- Special characters and node/field names: Should the X3D

specification put restrictions on a valid node name as used in DEF/USE? Besides preventing strange looking names it ensures that a scene graph can be exported to e.g. VRML syntax/encoding. Without any restrictions on such names we may be unable to generate a valid VRML file out of the scene. The reason is that an identifier in the VRML encoding has restrictions on which characters it may contain and there is no ways of escaping identifier characters. The problem is similar with names of custom fields such as for the Script node. For scripts in particular this raises another concern: If field names may include some punctuation characters the most scripting languages will be unable to refer to them. Entering such names will simply cause syntax errors. Also if non-ASCII characters are allowed, then the scripting engine needs to allow identifiers with non-ASCII characters and not all languages accept this.

- When a scene contains several ECMA scripts, do they live in entirely separate contexts (distinct SpiderMonkey runtimes), or do they live in a common name space, meaning that if two different scripts define functions with the same name then there is an error (a name clash). (Well, there are example scenes with two different scripts defining functions with the same name, but does this imply that they live in entirely distinct SpiderMonkey runtimes?)

- Does a custom input/output field of a scriptnode participate in event cascades when the url field of the Script node is empty?

- Will a script that is loaded as a result of assignment to the 'url' field of the Script node (and not during scene loading) get its 'initialize' method called? The question could be rephrased to: Is the intention of the 'initialize' method to initialize parts of the scene graph, or is it to initialize itself (i.e. variables that are internal to it), or both. Also it cannot be true what is stated in the X3D specification (October 22, 2002 - Part 1 - 29.2.3 Initialize() and shutdown() "'shutdown' shall be invoked when the corresponding Script node is deleted or the world containing the Script node is unloaded or replaced by another world." This is impossible if the the "url" field has been cleared or reassigned earlier.

- Is the use of the 'USE' attribute in the 'field' element of the XML encoding of X3D deprecated? If not, which has the highest precedence: A node mentioned in a USE attribute or a node element contained within the field element?

- When a custom script field is named "foo" and has access type 'inputOutput' may it then be accessed with the names "set_foo" and "foo_changed", or is this behavior restricted to conventional fields?

- If script source is put into the XML encoded X3D file as CDATA, must it always go into a CDATA section <![CDATA[...]]>, or is it the concatenation of all the CDATA pieces of the Script element?

- When both the "url" attribute occurs with contents and there is a CDATA section with contents in a Script element of the XML encoding of X3D, then does one take precedence over the other or is the CDATA appended/prepended to the list of alternative URIs? I would suggest that the CDATA is appended to the "url" vector, because it will always occur below the url-value in the file.

- How is the script language determined when it occurs as CDATA in the XML encoding of X3D? I expect that at least the default is ECMA. (it seems that a protocol like 'ecmascript:' may be specified as a prefix like in the 'url' field, but what if a protocol is not specified?)

- What is the full story about the encoding of MFString values occurring an an XML encoded X3D file? From the examples in the conformance suite it is obvious that both single-quote(') and double-quote(") have special meaning. It is also obvious that a pare of single quotes or a pare of double quotes delimit one string within the string sequence. BTW. how is a double-quote(") included in a string value in the VRML encoding? Is some sort of escaping defined in the VRML encoding? Is some sort of escaping defined in the XML encoding? Does single-quote(') and double-quote(") have special meanings in SFString also? If there exist a way of escaping characters then it this escaping mechanism active in other attribute values such as MFFloat?

- What should be the consequence of an empty value for a SFBool or an SFInt32 or an SFFloat etc. in the XML encoding? Error / true / false / default for field

- What should be the reaction to a comma(,) between two floats in the value of an MFFloat or an MFVec3f attribute in XML encoding? Error / ignore

- The X3D specification says that "The Text node does not participate in collision detection", but does this mean that one cannot activate a pointing device sensor via the text node, or does it only refer to the fact that the text node will not interfere with your avatar when walking?

- Why does the abstract X3D node type "ComposedGeometryNode" have the field named "coord" when that field is not used in the derived node type "ElevationGrid"? Maybe it should be removed from "ComposedGeometryNode" and then added to "IndexedFaceSet" and the other node types that need that field. I think it is rather ugly to have the "coord" field in an ElevationGrid when it has absolutely no function there.

- What does it mean that a certain viewpoint is bound? Is it a specific node that is bound, meaning that if that nodes name is changed then it is still that node which is bound. Or is it a specific node name that is bound, meaning that if the with that name gets anew name, then no viewpoint is bound hereafter. Also, this would imply that if the first viewpoint in a file has no name then it cannot be bound. Assuming that the binding of a viewpoint is an association with a specific node, then what happens if that node is removed from the scene graph? Is the binding then entirely broken or will it be reestablished if the node is put back into the scene graph?

- Specifying the name of the initial viewpoint with the URI does not necessarily specify a unique initial view, as the viewpoint with the specified name may be a child in two or more distinct transformations, in which case several different possible initial

viewpoint are selected. Its the same problem when some viewpoint node is bound due to receiving a set_bound event. Which occurrence of the node should be assumed? The specification mentions that 'The URL syntax ".../scene.wrl#ViewpointName" specifies the user's initial view when loading "scene.wrl" to be the first Viewpoint node in the X3D file that appears as DEF ViewpointName Viewpoint {...}.'. Taking this as a general rule, we should at each frame start out by searching through the scene graph for the first occurrence of the specified/bound view point. A relaxation of this rule could be that only initially we need to find the first occurrence, and then at each subsequent frame we only need to find SOME occurrence. The latter might enables us to track the viewpoint faster. The X3D specification says that the result of binding a viewpoint with multiple occurrences in the scene graph is undefined (Part 1 - 7.2.2 Bindable children nodes).

- For a viewpoint one can specify 'centerOfRotation'. This means that the center of rotation need not be at the center of the screen/view. I think this is inconvenient since the track ball navigation only works well if the center of rotation is centered on the screen. Thus it would have been better if one specified the depth of rotation instead. I see two solutions to support the "center of rotation" as specified by X3D. One is to have the virtual ball in the center as usual but actually rotate around the specified point, keeping the distance to the center of rotation and the trackball radius equal. This has the unwanted effect that the center of rotation is not coincident with the center of the trackball. The other solution is to displace the ball to make the ball center coincident with the center of rotation, but then the user wont have an intuitive feeling of the ball surface as is usually the case.

- How does Browser::endUpdate handle errors, that is, violations like writing to a private field of a realized node. Note that the node may get realized due to an earlier event in the same buffer, and this condition can henceforth not be detected any earlier that at the call to endUpdate. Yet the SAI specification does not mention any other sources of error than BROWSER_DISPOSED.

- Shouldn't getExecutionContext have been a node service? The description seems to fit much more if that was the case. Part 2 - 6.3.10.

- The specification of the SAI specifies that node operations should raise InvalidNodeException after dispose, but this is only possible if node servants are kept active forever. This is hardly reasonable in cases where the application has removed a node from the scene graph and then calls dispose on it afterwards. If the node and its associated servant (CORBA or the like) is never deallocated it would result in memory leak.

- What does it mean when a node is disposed through the SAI. Is the client able to reacquire a reference to that same node?

- When SAI.Browser.getNode is called twice for the same name, must it be the same reference that is returned from both calls?

- Does a node acquired through the SAI represent the node at the server or a reference to it?

```
- Shouldn't X3DUrlObject and X3DBoundedObject be descendants of X3DNode.



FAQ comprised of answered questions:
====================================

Q When an ElevationGrid is unlit but specifies colors through a
  ColorNode, does the ElevationGrid become completely white or does the
  colors from the ColorNode still produce a multicolored ElevationGrid
  even when it is unlit? What about PointSets and IndexedLineSets can
  they have colors different from white when the PointSet is unlit
  (that is, there is either no AppearanceNode or no MaterialNode)?

A No - the 'unlit' condition allows both texturing and coloring
  through use of ColorNodes. It only disables lighting based on
  enabled light sources and material properties. PointSets and
  IndexedLineSets are never lit but may have colors independent of
  whether the shape is lit or unlit. See X3D specification of October
  22, 2002, Part 1 - 17.2.2.2 Lighting 'off'.
```

Appendix F

User Guide

To get a short help message and a list of all available switches, their default value and a short description of each switch, please write the following command:

```
# 3dconsole --help
```

3D-Console will write out a short help message and list all the switches.

```
3D-Console by Brian Kristiansen & Kristian Spangsege

Synopsis: 3dconsole URI

The URI must refer to an X3D file and
may be specified relative to the current working directory.
If you specify a fragment identifier it will be interpreted
as the name of the viewpoint to use. For example
  3dconsole house.x3d#front
will render the house from the viewpoint named 'front'.

Available options:
-h, --help              yes           Describe the parameters
-c, --config-file       ...chon.conf  Set the path of the configuration file
--save-config           no            Save the modified configuration into the
                                      configuration file
-F, --full-screen       no            View in fullscreen mode
-r, --resolution        1000x1000     Set resolution
-B, --background-color  0,0,0,1       R,G,B or R,G,B,A color of background
                                      (range 0-1)
-e, --eyepoint          0,0,10        Location of the eye in the default view
-l, --lookat            0,0,0         Point of interest in the default view
```



```
-v, --field-of-view     0.785398      Smallest edge-to-edge angle of viewplane
                                      when seen from the eye
-H, --head-light        yes           Enable head light
-L,                     no            Display light sources as points in their
--show-light-sources                  color
-w, --wireframe         no            Render the scene in wireframe instead of
                                      the usual filled polygons
-t, --texture           yes           Enable use of texture mapping
-m, --mipmap            no            Enable mipmaped textures. You need this
                                      for textures whos dimentions are not a
                                      power of 2. Note also that there are some
                                      probplems when using mipmaps together with
                                      projective textures.
-N, --show-normals      no            Reveal the normal at each rendered vertex
-f, --frame-rate        40            Set the number of frames per second
-s, --subdivision       25x25         Set subdivision of curved geometry such as
                                      sphers, tori, cylinders and cones
-b,                     1.5           Set the delay in seconds between pings
--application-heartbea                from the server to the applications
t
-p,                     -1            The TCP/IP port on which the CORBA Orb
--corba-endpoint-port                 listens (-1 for auto assign)
-a, --accept-clients    no            Accept connection from client applications
                                      via the SAI
-n, --console-name      archon        Set the name of this server as known by
                                      SAI clients
-D, --dump-scene        no            Write out a textual description of the
                                      scene at the time of exit
-P, --progressive-load  no            View scene while loading
--font-path             ...ult/Type1  A colon seperated list of directories in
                                      which to look for fonts. Empty entries are
                                      ignored.
```

To use 3D-Console as an X3D viewer while not accepting connections from applications, use the following command:

```
# 3dconsole http://www.web3d.org/TaskGroups/x3d\
/translation/examples/Conformance/Geometry\
/Sphere/default.x3d
```

To view an X3D file from the filesystem, use the following command instead:

```
# 3dconsole /tmp/default.x3d
```

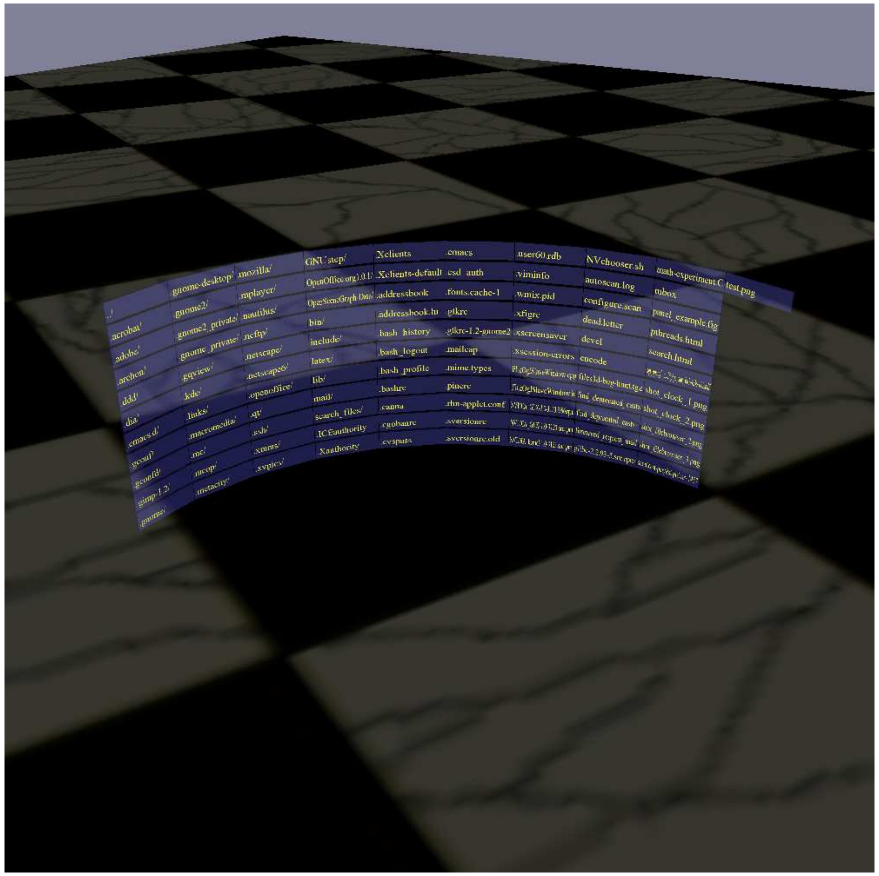
The 3D-Console will open a window and display the X3D scene from the specified file.

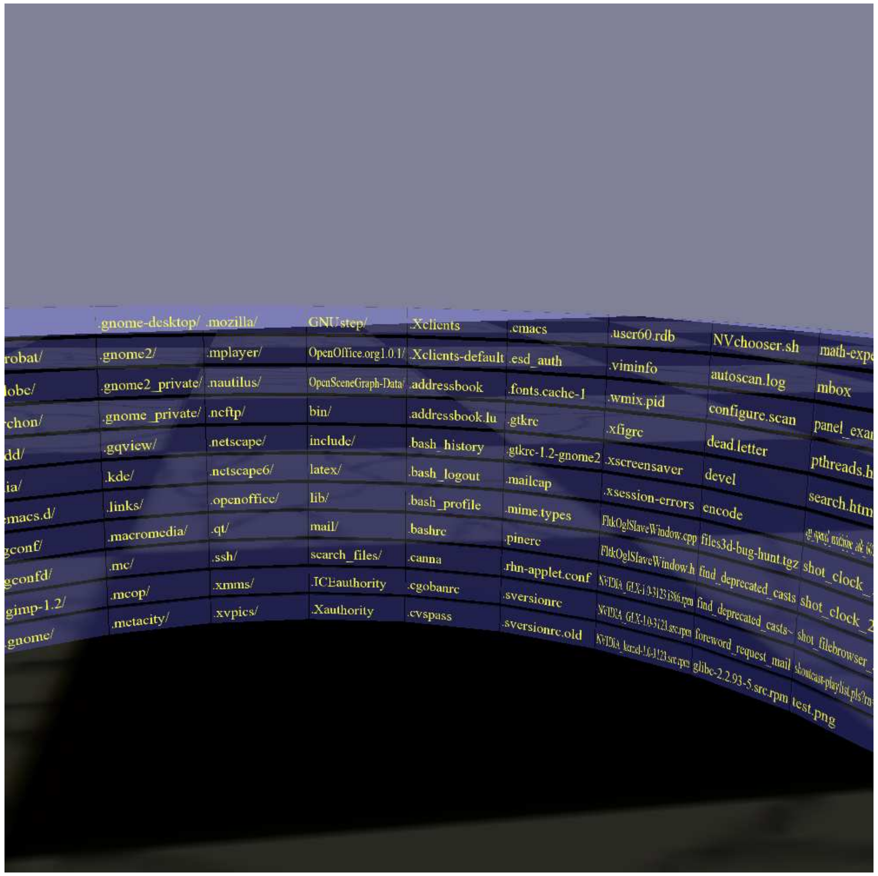
For opening a default X3D scene (named scene.x3d) with textures using mipmaps, while accepting connections from applications, please use the following command:

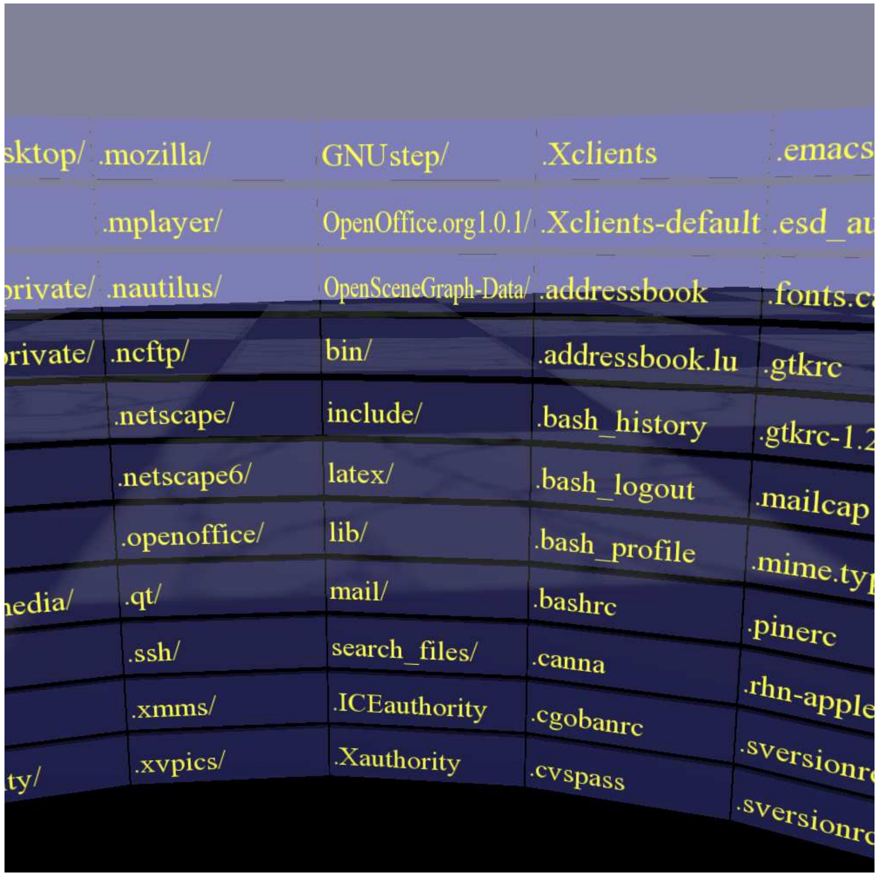
```
# 3dconsole /tmp/scene.x3d -a -m
```

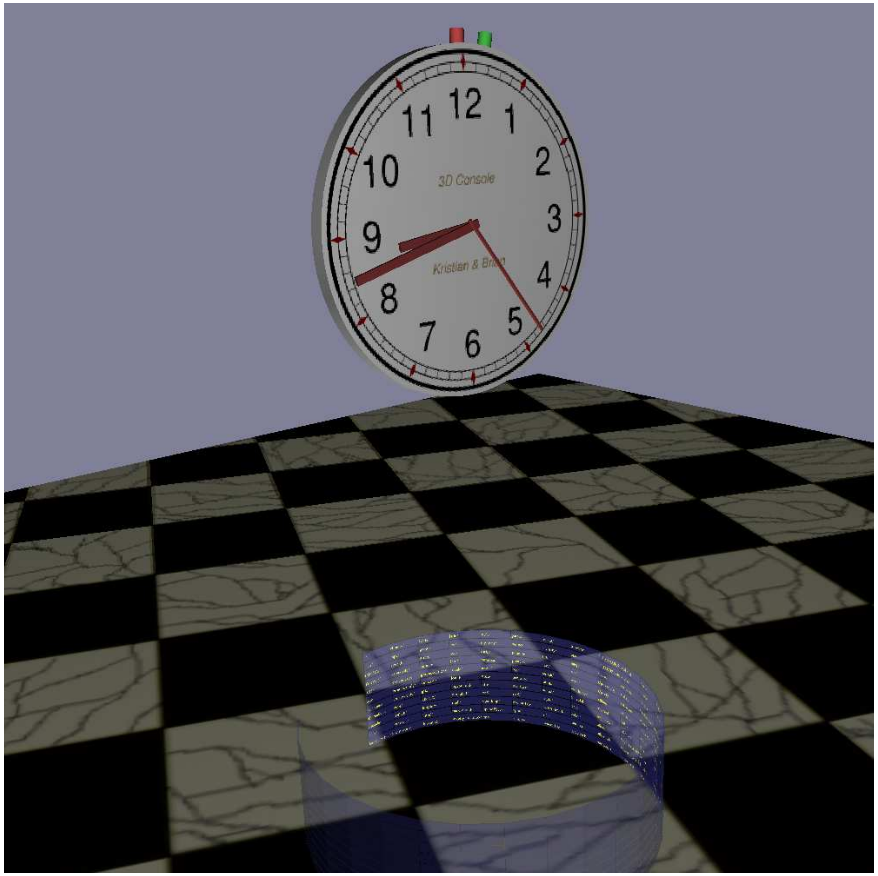
Appendix G

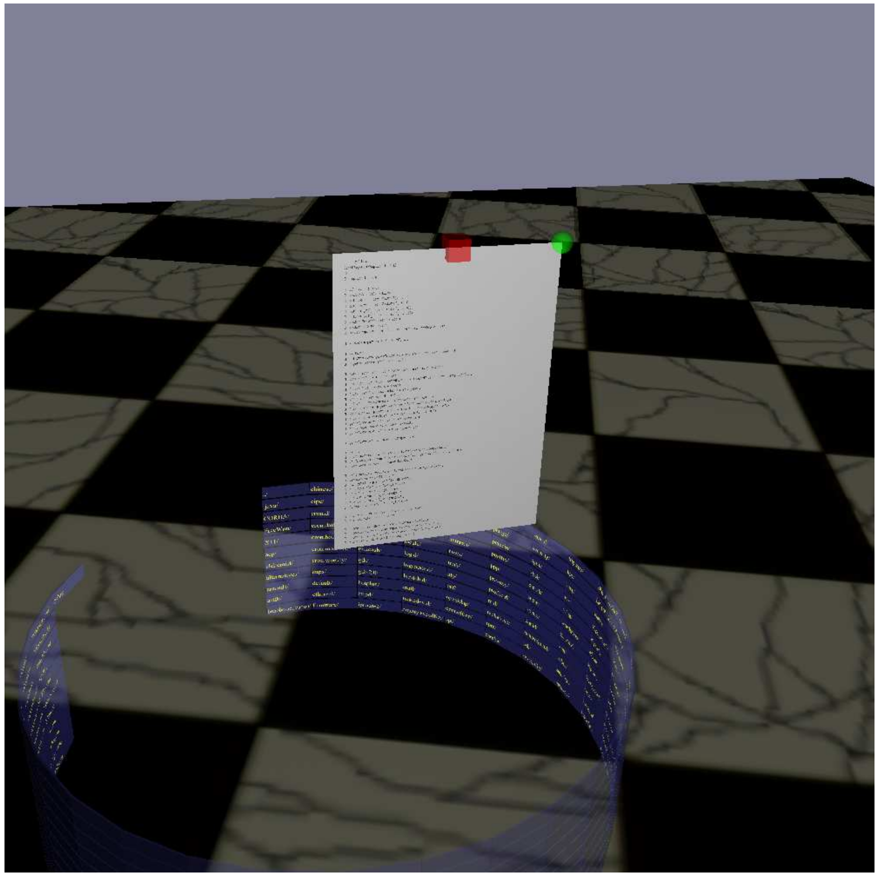
Screen-shots

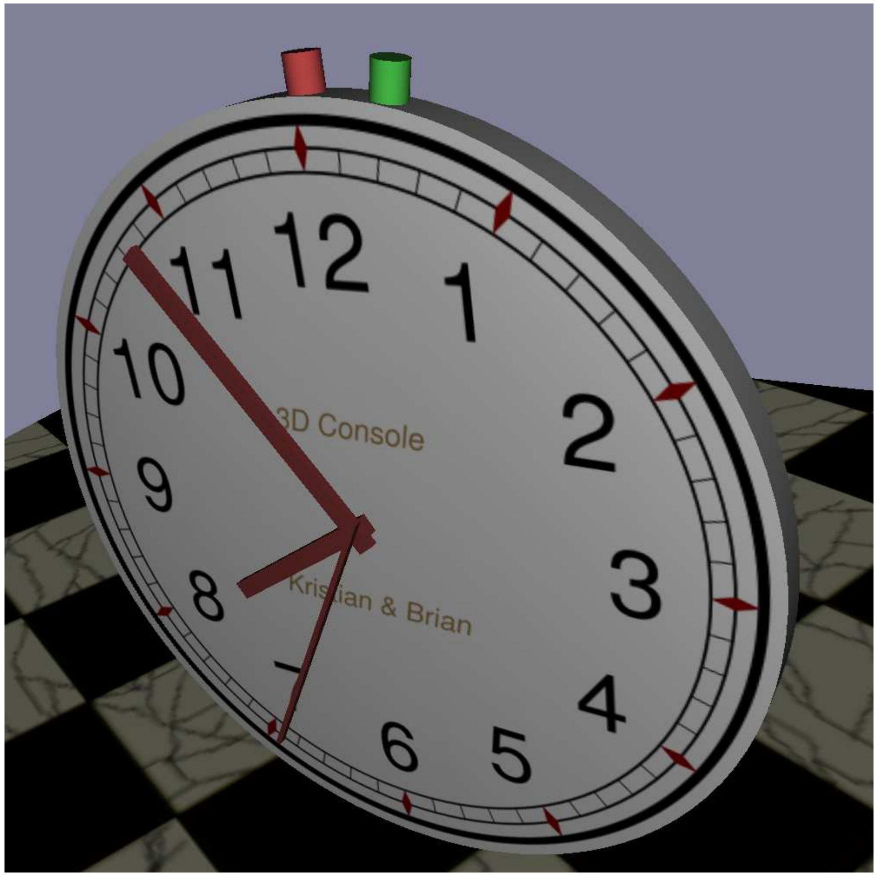


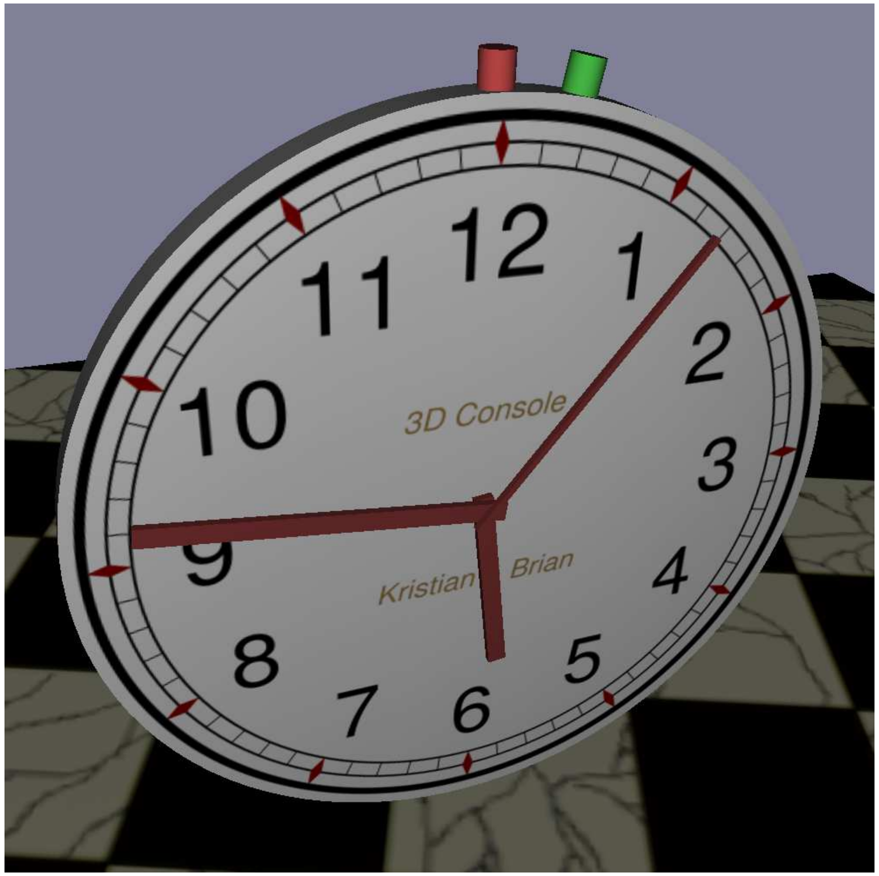

.gnome-desktop/
.mozilla/
GNUstep/
.Xclients
.emacs
.user60.rdb
NVchooser.sh
.gnome2/
.mplayer/
OpenOffice.org1.0.1/
.Xclients-default
.esd_auth
.viminfo
autoscan.log
mbox
.gnome2_private/
.nautilus/
OpenSceneGraph-Data/
.addressbook
.fonts.cache-1
.wmix.pid
configure.scan
.gnome_private/
.ncftp/
bin/
.addressbook.lu
.gtkrc
.xfigrc
dead.letter
.gqview/
.netscape/
include/
.bash_history
.gtkrc-1.2-gnome2
.xscreensaver
devel
.kde/
.netscape6/
latex/
.bash_logout
.mailcap
.xsession-errors
encode
.links/
.openoffice/
lib/
.bash_profile
.mime.types
.macromedia/
.qt/
mail/
.bashrc
.pinerc
.mc/
.ssh/
search_files/
.canna
.rhn-applet.conf
.mcop/
.xmms/
.ICEauthority
.cgobanrc
.sversionrc
.metacity/
.xvpics/
.Xauthority
.cvspass
.sversionrc.old
files3d-bug-hunt.tgz
test.png

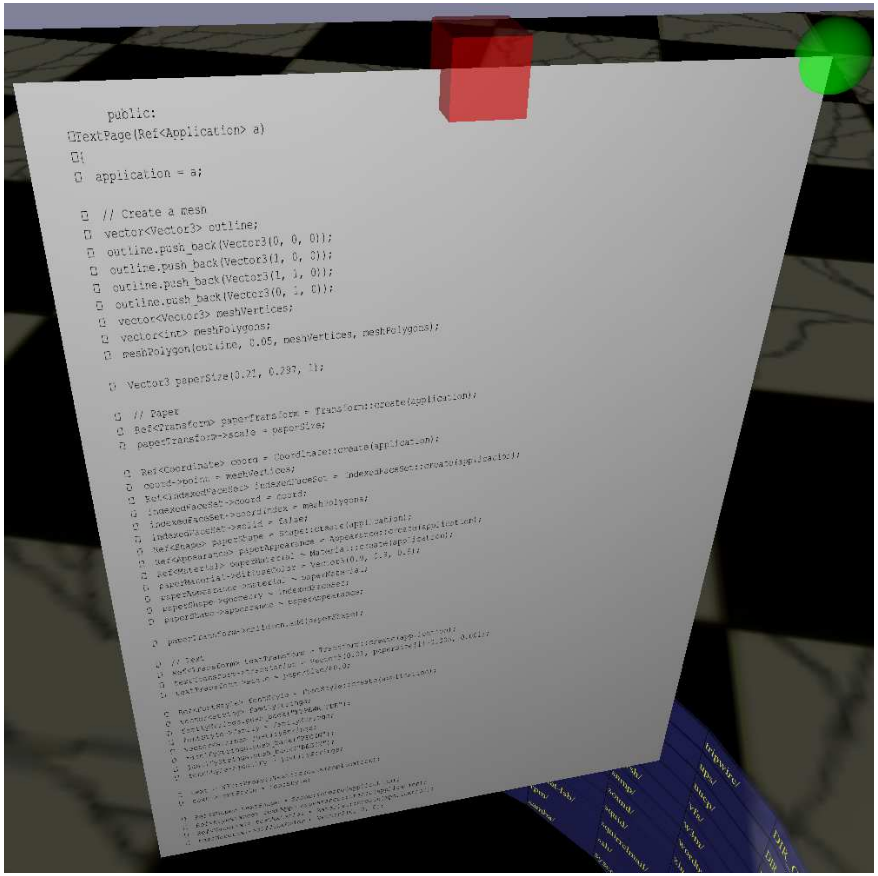

.mozilla/
GNUstep/
.Xclients
.mplayer/
OpenOffice.org1.0.1/
.Xclients-default
.nautilus/
OpenSceneGraph-Data/
.addressbook
.ncftp/
bin/
.addressbook.lu
.gtkrc
.netscape/
include/
.bash_history
.netscape6/
latex/
.bash_logout
.mailcap
.openoffice/
lib/
.bash_profile
.qt/
mail/
.bashrc
.pinerc
.ssh/
search_files/
.canna
.xmms/
.ICEauthority
.cgobanrc
.xvpics/
.Xauthority
.cvspass

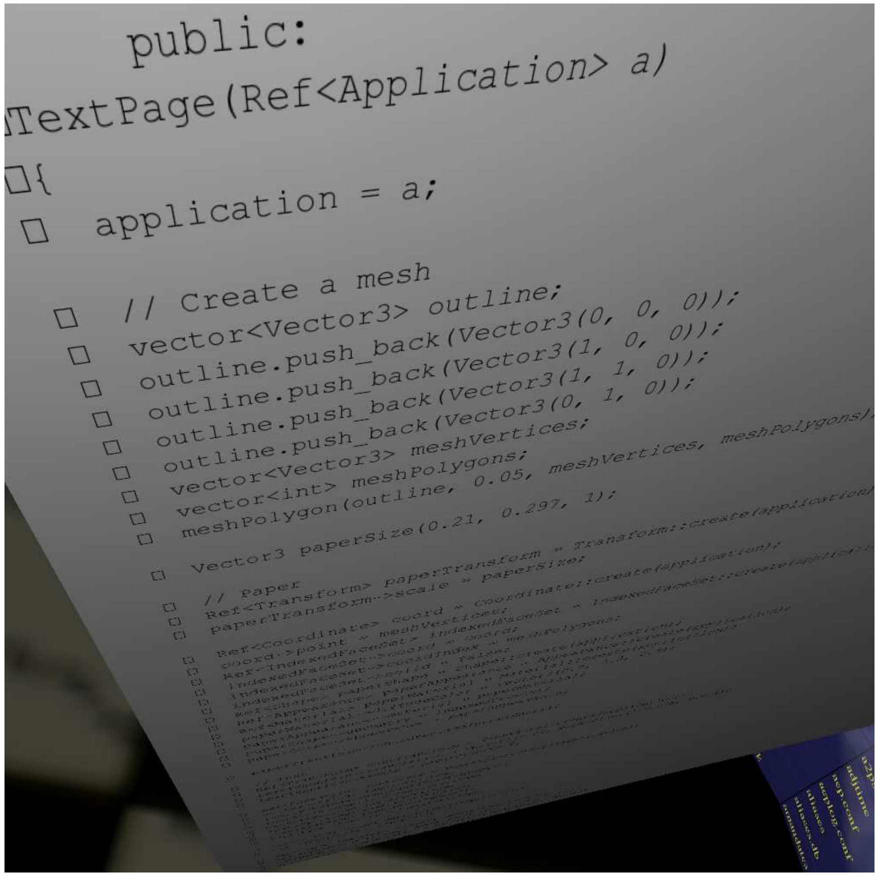

3D Console

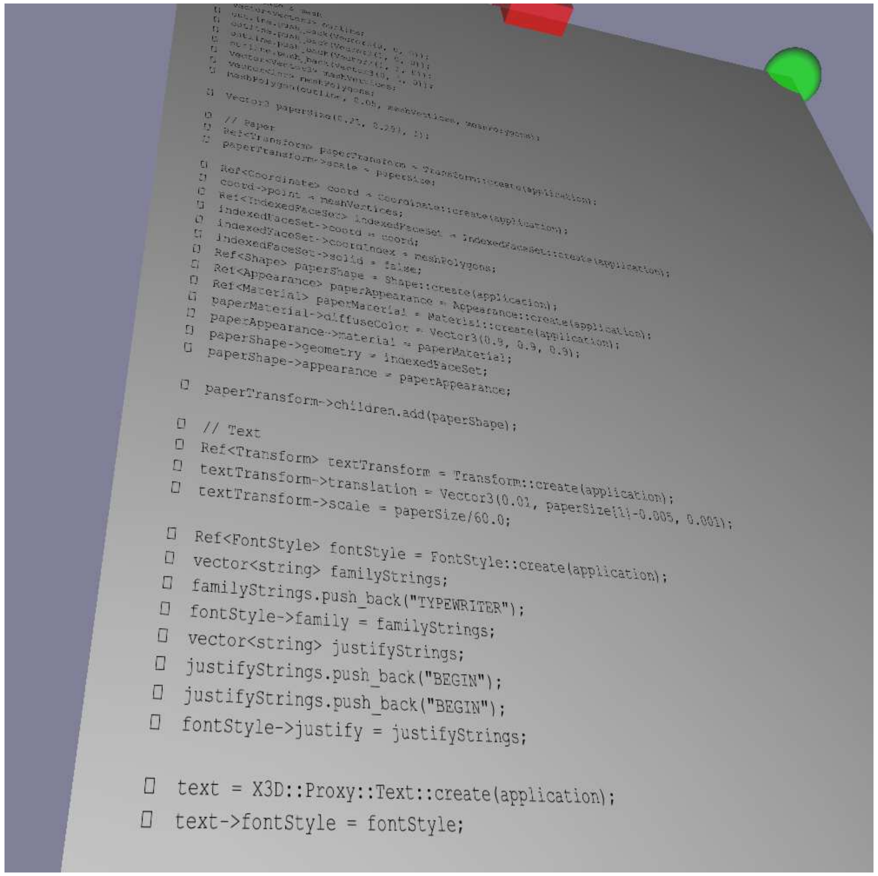



3D Console
Kristian & Brian

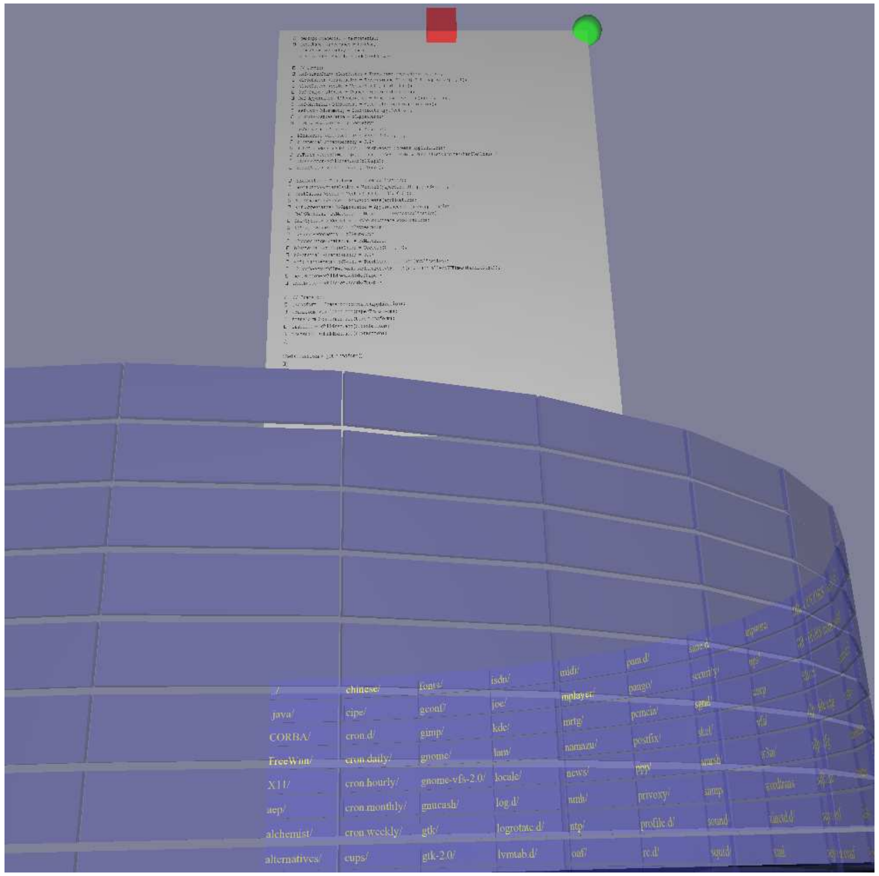

3D Console
Kristian
Brian


public:
TextPage(Ref<Application> a)
{
application = a;
// Create a mesh
vector<Vector3> outline;
outline.push_back(Vector3(0, 0, 0));
outline.push_back(Vector3(1, 0, 0));
outline.push_back(Vector3(1, 1, 0));
outline.push_back(Vector3(0, 1, 0));
vector<Vector3> meshVertices;
vector<int> meshPolygons;
meshPolygon(outline, 0.05, meshVertices, meshPolygons);
Vector3 paperSize(0.21, 0.297, 1);
// Paper


public:
TextPage(Ref<Application> a)
{
application = a;
// Create a mesh
vector<Vector3> outline;
outline.push_back(Vector3(0, 0, 0));
outline.push_back(Vector3(1, 0, 0));
outline.push_back(Vector3(1, 1, 0));
outline.push_back(Vector3(0, 1, 0));
vector<Vector3> meshVertices;
vector<int> meshPolygons;
meshPolygon(outline, 0.05, meshVertices, meshPolygons);
Vector3 paperSize(0.21, 0.297, 1);
// Paper
Ref<Transform> paperTransform = Transform::create(application);
paperTransform->scale = paperSize;



```
Ref<Coordinate> coord = Coordinate::create(application);
coord->point = meshVertices;
Ref<IndexedFaceSet> indexedFaceSet = IndexedFaceSet::create(application);
indexedFaceSet->coord = coord;
indexedFaceSet->coordIndex = meshPolygons;
indexedFaceSet->solid = false;
Ref<Shape> paperShape = Shape::create(application);
Ref<Appearance> paperAppearance = Appearance::create(application);
Ref<Material> paperMaterial = Material::create(application);
paperMaterial->diffuseColor = Vector3(0.9, 0.9, 0.9);
paperAppearance->material = paperMaterial;
paperShape->geometry = indexedFaceSet;
paperShape->appearance = paperAppearance;
paperTransform->children.add(paperShape);
// Text
Ref<Transform> textTransform = Transform::create(application);
textTransform->translation = Vector3(0.01, paperSize[1]-0.005, 0.001);
textTransform->scale = paperSize/60.0;
Ref<FontStyle> fontStyle = FontStyle::create(application);
vector<string> familyStrings;
familyStrings.push_back("TYPEWRITER");
fontStyle->family = familyStrings;
vector<string> justifyStrings;
justifyStrings.push_back("BEGIN");
justifyStrings.push_back("BEGIN");
fontStyle->justify = justifyStrings;
text = X3D::Proxy::Text::create(application);
text->fontStyle = fontStyle;
```



chinese/
java/
CORBA/
FreeWnn/
X11/
alchemist/
alternatives/
cipe/
cron.d/
cron.daily/
cron.hourly/
cron.monthly/
cron.weekly/
cups/
gconf/
gimp/
gnome/
gnome-vfs-2.0/
gnucash/
gtk/
gtk-2.0/
locale/
log.d/
logrotate.d/

Appendix H

CORBA IDL for client API

```
module x3d
{
  module sai
  {
    typedef sequence<char> chararray;

    exception ServerErrorException
    {
      string message;
    };

    exception ApplicationErrorException
    {
      string message;
    };

    exception FileRequestException
    {
      string message;
    };

    exception StreamReadException
    {
      string message;
    };

    interface StreamReader
    {
      chararray read(in long n) raises(StreamReadException);
      void dispose();
    };

    typedef unsigned long     ulong;

    typedef sequence<boolean> booleanarray;
```



```
    typedef sequence<long>    longarray;
    typedef sequence<float>   floatarray;
    typedef sequence<double>  doublearray;
    typedef sequence<string>  stringarray;
    typedef sequence<ulong>   ulongarray;

    typedef sequence<doublearray> doubledoublearray;
    typedef sequence<floatarray>  floatdoublearray;
    typedef sequence<longarray>   longdoublearray;

    exception InvalidServerException {};            // Server reference
    exception InvalidApplicationException {};       // Application reference
    exception InvalidSessionException {};           // Session reference
    exception InvalidExecutionContextException {}; // Context ID
    exception InvalidNodeException {};              // Node ID
    exception InvalidFieldException {};             // Field ID
    exception InvalidAccessTypeException {};        // Field input output capablility
    exception InvalidOperationTimingException {};  // Node life cycle
    exception InvalidIndexException {};             // Sequence value subscription
    exception InvalidNameException {};              // Node type name of field name
    exception InvalidTypeException {};              // Node type ID
    exception InvalidX3DException {};               // Syntax error in X3D source text
    exception NodeInUseException {};                // Context relation conflict
    exception AlreadyLaunchedException {};          // Application context launch

    /**
     * Is thrown when a Value object is not valid. That is, if
     * Value.val does not hold something of the type specified by
     * Value.type.
     */
    exception InvalidValueException {};

    struct Value
    {
      any val;
      ulong type;
    };

    struct Event
    {
      Value val;
      ulong cookie;
    };

    typedef sequence<Event> EventSeq;

    struct DisposeEntry
    {
      ulong id;
      ulong transCount;
    };

    typedef sequence<DisposeEntry> DisposeSeq;

    interface Session
    {
      // Session (or Browser) services
```

```
string getServerName()
  raises(ServerErrorException, InvalidSessionException);

double getCurrentFrameRate()
  raises(ServerErrorException, InvalidSessionException);

ulong createApplicationScene(in string baseUri,
                             in boolean localFileUri)
  raises(ServerErrorException, InvalidSessionException);

ulong loadApplicationScene(in string uri,
                           in boolean localFileUri)
  raises(ServerErrorException, InvalidSessionException,
         FileRequestException, InvalidX3DException);

void launchApplicationScene(in ulong contextId)
  raises(ServerErrorException, InvalidSessionException,
         InvalidExecutionContextException, AlreadyLaunchedException);

/**
 * Enter the "buffered changes" mode.
 */
void beginUpdate()
  raises(ServerErrorException, InvalidSessionException);

/**
 * Commit the buffered changes and end the "buffered changes
 * mode".
 */
void endUpdate()
  raises(ServerErrorException, InvalidSessionException);

void print(in string message)
  raises(ServerErrorException, InvalidSessionException);

/**
 * Enable or disable "synchronized changes" mode.  If enabled,
 * then whenever the session is not in "buffered changes" mode
 * then each change is applied before the change request returns
 * to the caller and the caller is notified if there was an
 * access error. If disabled, then changes are simply queued up
 * and applied whenever the server finds it appropriate. This
 * means that access errors will not get reported back to the
 * caller. The default is "enabled".
 */
void setSynchronizedChangesEnabled(in boolean enabled)
  raises(ServerErrorException, InvalidSessionException);

void disposeBatch(in DisposeSeq contexts, in DisposeSeq nodes)
  raises(ServerErrorException, InvalidSessionException);

void disposeSession()
  raises(ServerErrorException);


// ExecutionContext services

string getBaseUri(in ulong contextId)
```



```
  raises(ServerErrorException, InvalidSessionException,
         InvalidExecutionContextException);

ulong getRootGroup(in ulong contextId)
  raises(ServerErrorException, InvalidSessionException,
         InvalidExecutionContextException);

void setRootGroup(in ulong contextId, in ulong nodeId)
  raises(ServerErrorException, InvalidSessionException,
         InvalidExecutionContextException, InvalidNodeException);

ulong createNode(in ulong contextId, in string typeName)
  raises(ServerErrorException, InvalidSessionException,
         InvalidExecutionContextException, InvalidNameException);

ulong createNodeFast(in ulong contextId, in ulong typeId)
  raises(ServerErrorException, InvalidSessionException,
         InvalidExecutionContextException, InvalidTypeException);

ulong getNode(in ulong contextId, in string nodeName)
  raises(ServerErrorException, InvalidSessionException,
         InvalidExecutionContextException, InvalidNameException);

void addRoute(in ulong contextId,
              in ulong fromNodeId, in ulong fromFieldId,
              in ulong toNodeId, in ulong toFieldId)
  raises(ServerErrorException, InvalidSessionException,
         InvalidExecutionContextException, InvalidNodeException,
         InvalidFieldException, InvalidAccessTypeException);

void delRoute(in ulong contextId,
              in ulong fromNodeId, in ulong fromFieldId,
              in ulong toNodeId, in ulong toFieldId)
  raises(ServerErrorException, InvalidSessionException,
         InvalidExecutionContextException, InvalidNodeException,
         InvalidFieldException, InvalidAccessTypeException);

ulong createGroupFromStream(in ulong contextId, in StreamReader r)
  raises(ServerErrorException, InvalidSessionException,
         InvalidExecutionContextException, InvalidX3DException);

void disposeContext(in ulong contextId)
  raises(ServerErrorException, InvalidSessionException);


// Node services

string getNodeType(in ulong nodeId)
  raises(ServerErrorException, InvalidSessionException,
         InvalidNodeException);

ulong getField(in ulong nodeId, in string name)
  raises(ServerErrorException, InvalidSessionException,
         InvalidNodeException, InvalidNameException);

void disposeNode(in ulong nodeId)
  raises(ServerErrorException, InvalidSessionException);
```

```
// Field services

ulong getFieldType(in ulong nodeId, in ulong fieldId)
  raises(ServerErrorException, InvalidSessionException,
         InvalidNodeException, InvalidFieldException);

string getFieldName(in ulong nodeId, in ulong fieldId)
  raises(ServerErrorException, InvalidSessionException,
         InvalidNodeException, InvalidFieldException);

/*
 * 0 -> initializeOnly
 * 1 -> inputOnly
 * 2 -> outputOnly
 * 3 -> inputOutput
 */
ulong getFieldAccessType(in ulong nodeId, in ulong fieldId)
  raises(ServerErrorException, InvalidSessionException,
         InvalidNodeException, InvalidFieldException);

void registerFieldInterest(in ulong nodeId, in ulong fieldId,
                           in ulong cookie)
  raises(ServerErrorException, InvalidSessionException,
         InvalidNodeException, InvalidFieldException,
         InvalidAccessTypeException);

void unRegisterFieldInterest(in ulong nodeId, in ulong fieldId,
                             in ulong cookie)
  raises(ServerErrorException, InvalidSessionException,
         InvalidNodeException, InvalidFieldException,
         InvalidAccessTypeException);

Value getFieldValue(in ulong nodeId, in ulong fieldId)
  raises(ServerErrorException, InvalidSessionException,
         InvalidNodeException, InvalidFieldException,
         InvalidAccessTypeException, InvalidOperationTimingException);

void setFieldValue(in ulong nodeId, in ulong fieldId, in Value v)
  raises(ServerErrorException, InvalidSessionException,
         InvalidNodeException, InvalidFieldException,
         InvalidAccessTypeException, InvalidOperationTimingException,
         InvalidValueException, NodeInUseException);

void disposeField(in ulong fieldId)
  raises(ServerErrorException, InvalidSessionException);


// SequenceField services

Value getFieldValueAt(in ulong nodeId, in ulong fieldId, in ulong index)
  raises(ServerErrorException, InvalidSessionException,
         InvalidNodeException, InvalidFieldException,
         InvalidAccessTypeException, InvalidOperationTimingException,
         InvalidIndexException);

void setFieldValueAt(in ulong nodeId, in ulong fieldId, in ulong index,
```



```
                         in Value v)
      raises(ServerErrorException, InvalidSessionException,
             InvalidNodeException, InvalidFieldException,
             InvalidAccessTypeException, InvalidOperationTimingException,
             InvalidIndexException, InvalidValueException,
             NodeInUseException);

    void setFieldValueAdd(in ulong nodeId, in ulong fieldId, in Value v)
      raises(ServerErrorException, InvalidSessionException,
             InvalidNodeException, InvalidFieldException,
             InvalidAccessTypeException, InvalidOperationTimingException,
             InvalidValueException, NodeInUseException);

    void setFieldValueDel(in ulong nodeId, in ulong fieldId, in Value v)
      raises(ServerErrorException, InvalidSessionException,
             InvalidNodeException, InvalidFieldException,
             InvalidAccessTypeException, InvalidOperationTimingException,
             InvalidValueException, NodeInUseException);
  };

  interface Application
  {
    void handleEvent(in EventSeq s);
    StreamReader requestFile(in string path) raises(FileRequestException);
    void log(in string message);
  };

  interface Server
  {
    ulong getNodeTypeBitsInId()
      raises(ServerErrorException, InvalidServerException);

    ulong getNodeType(in string typeName)
      raises(ServerErrorException, InvalidServerException,
             InvalidNameException);

    ulong getImmediateField(in ulong nodeTypeId, in string fieldName)
      raises(ServerErrorException, InvalidServerException,
             InvalidTypeException, InvalidNameException);

    Session createSession(in Application a)
      raises(ServerErrorException, InvalidServerException);
  };
 };
};
```

Appendix I

Source code

Listing: **archon/sai_test_apps/clock/main.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#include <iostream>

#include <archon/math/vector.H>
#include <archon/util/time.H>
#include <archon/util/mutex.H>
#include <archon/util/ref.H>
#include <archon/util/options.H>

#include <archon/x3d/proxy/proxy.H>

namespace Archon
{
  namespace SaiTestApps
  {
    namespace Clock
    {
      using namespace std;
      using namespace Math;
      using namespace Utilities;
      using namespace X3D::Proxy;

      static bool quit = false;
```

```
static Ref<Material> quitMaterial;

static bool fastForward = false;
static Ref<Material> fastForwardMaterial;

void quitIsOver(const bool v)
{
  quitMaterial->diffuseColor = (v ? Vector3(1, 0, 0) : Vector3
       (0.8, 0.3, 0.3));
}

void quitTouch(const Time)
{
  quit = true;
}

void fastForwardIsOver(const bool v)
{
  fastForwardMaterial->diffuseColor = (v ? Vector3(0, 1, 0) : Vector3
       (0.3, 0.8, 0.3));
}

void fastForwardTouch(const Time)
{
  fastForward = !fastForward;
}

int main(int argc, const char *argv[])
{
  bool   opt_help              = false;
  int    opt_corbaEndpointPort = -1; // Auto assign
  string opt_serverName        = "archon";

  Options o;

  o.addSwitch("h", "help", opt_help, true,
           "Describe the parameters");
  o.addConfig("p", "corba-endpoint-port", opt_corbaEndpointPort,
       opt_corbaEndpointPort,
           "The TCP/IP port on which the CORBA Orb listens (-1 for auto
                assign)",
           Options::wantArg_always);
  o.addConfig("n", "server-name", opt_serverName, opt_serverName,
           "The name of the server to connect to",
           Options::wantArg_always);

  if(o.processCommandLine(argc, argv))
  {
    cerr << "Try --help\n";
    return 1;
  }

  if(opt_help)
  {
    cout <<
      "Test Application for 3D-Console by Brian Kristiansen & Kristian
           Spangsege\n"
      "\n"
```



```
      "Synopsis: " << argv[0] << "\n"
      "\n"
      "Available options:\n";
    cout << o.list();
    return 0;
  }

  if(argc > 1)
  {
    cerr << "Too many aguments\n";
    cerr << "Try --help\n";
    return 1;
  }

  Ref<Session> session =
    Session::create(opt_serverName, opt_corbaEndpointPort);

  Ref<Application> application =
    Application::create(session, Uri());

  session->beginUpdate();

  Ref<Group> rootGroup = Group::create(application);

  Ref<Transform> rootTransform = Transform::create(application);
  rootTransform->rotation = AxisRot3(Vector3(1, 0, 0), M_PI/2);
  rootTransform->scale = Vector3(25, 25, 25);
  rootTransform->translation = Vector3(0, 100, 0);
  rootGroup->children.add(rootTransform);

  {
    Ref<Cylinder> cyl = Cylinder::create(application);
    cyl->side = false;
    cyl->top = true;
    cyl->bottom = false;
    cyl->height = 0.1;
    Ref<Shape> shape = Shape::create(application);
    shape->geometry = cyl;
    Ref<Appearance> apperance = Appearance::create(application);
    apperance->material = Material::create(application);
    Ref<ImageTexture> texture = ImageTexture::create(application);
    texture->url.add("clock.png");
    apperance->texture = texture;
    shape->appearance = apperance;
    rootTransform->children.add(shape);
  }

  {
    Ref<Cylinder> cyl = Cylinder::create(application);
    cyl->side = true;
    cyl->top = false;
    cyl->bottom = true;
    cyl->height = 0.1;
    Ref<Shape> shape = Shape::create(application);
    shape->geometry = cyl;
    Ref<Appearance> apperance = Appearance::create(application);
    apperance->material = Material::create(application);
    shape->appearance = apperance;
```

```
  rootTransform->children.add(shape);
}

// Quit
{
  Ref<Cylinder> cyl = Cylinder::create(application);
  cyl->side = true;
  cyl->top = true;
  cyl->bottom = false;
  cyl->height = 0.1;
  cyl->radius = 0.04;
  Ref<Shape> shape = Shape::create(application);
  shape->geometry = cyl;
  Ref<Appearance> apperance = Appearance::create(application);
  quitMaterial = Material::create(application);
  quitMaterial->diffuseColor = Vector3(0.8, 0.3, 0.3);
  apperance->material = quitMaterial;
  shape->appearance = apperance;
  Ref<TouchSensor> touch = TouchSensor::create(application);
  touch->isOver.registerInterest<SFBool>(new FuncCaller<SFBool>(
       quitIsOver));
  touch->touchTime.registerInterest<SFTime>(new FuncCaller<SFTime>(
       quitTouch));
  Ref<Transform> transform = Transform::create(application);
  transform->children.add(shape);
  transform->children.add(touch);
  transform->translation = Vector3(0, 0, -1.03);
  transform->rotation = AxisRot3(Vector3(-1, 0, 0), M_PI/2);
  rootTransform->children.add(transform);
}

// Fast forward
{
  Ref<Cylinder> cyl = Cylinder::create(application);
  cyl->side = true;
  cyl->top = true;
  cyl->bottom = false;
  cyl->height = 0.1;
  cyl->radius = 0.04;
  Ref<Shape> shape = Shape::create(application);
  shape->geometry = cyl;
  Ref<Appearance> apperance = Appearance::create(application);
  fastForwardMaterial = Material::create(application);
  fastForwardMaterial->diffuseColor = Vector3(0.3, 0.8, 0.3);
  apperance->material = fastForwardMaterial;
  shape->appearance = apperance;
  Ref<TouchSensor> touch = TouchSensor::create(application);
  touch->isOver.registerInterest<SFBool>(new FuncCaller<SFBool>(
       fastForwardIsOver));
  touch->touchTime.registerInterest<SFTime>(new FuncCaller<SFTime>(
       fastForwardTouch));
  Ref<Transform> subTransform = Transform::create(application);
  subTransform->children.add(shape);
  subTransform->children.add(touch);
  subTransform->translation = Vector3(0, 0, -1.03);
  subTransform->rotation = AxisRot3(Vector3(-1, 0, 0), M_PI/2);
  Ref<Transform> transform = Transform::create(application);
  transform->children.add(subTransform);
```



```
  transform->rotation = AxisRot3(Vector3(0, -1, 0), M_PI/16);
  rootTransform->children.add(transform);
}

Ref<ChildNode> pointer;
{
  Ref<Box> box = Box::create(application);
  box->size = Vector3(0.05, 0.02, 0.9);
  Ref<Shape> shape = Shape::create(application);
  shape->geometry = box;
  Ref<Appearance> apperance = Appearance::create(application);
  Ref<Material> material = Material::create(application);
  material->diffuseColor = Vector3(0.5, 0.2, 0.2);
  apperance->material = material;
  shape->appearance = apperance;
  Ref<Transform> transform = Transform::create(application);
  transform->children.add(shape);
  transform->translation = Vector3(0, 0, -0.4);
  pointer = transform;
}

Ref<Transform> seconds = Transform::create(application);
seconds->children.add(pointer);
seconds->translation = Vector3(0, 0.1, 0);
seconds->scale = Vector3(0.3, 1, 1);
rootTransform->children.add(seconds);

Ref<Transform> minutes = Transform::create(application);
minutes->children.add(pointer);
minutes->translation = Vector3(0, 0.08, 0);
rootTransform->children.add(minutes);

Ref<Transform> hours = Transform::create(application);
hours->children.add(pointer);
hours->translation = Vector3(0, 0.06, 0);
hours->scale = Vector3(1, 1, 0.6);
rootTransform->children.add(hours);

session->endUpdate();

application->setRootGroup(rootGroup);
application->launch();

long addition = 0;

Archon::Utilities::Time delay(0, 100000000l);
while(!quit)
{
  long sec, nsec;
  Time t = Time::now();
  if(fastForward) addition += 60000;
  t.addMilliSeconds(addition);
  t.get(sec, nsec);
  time_t t1 = sec;
  tm t2;
  localtime_r(&t1, &t2);

  double n = nsec/1E9;
```

```
          double s = (t2.tm_sec + n)/60;
          double m = (t2.tm_min + s)/60;
          double h = (t2.tm_hour + m)/12;

          session->beginUpdate();

          seconds->rotation = AxisRot3(Vector3(0, -1, 0), s*2*M_PI);
          minutes->rotation = AxisRot3(Vector3(0, -1, 0), m*2*M_PI);
          hours->rotation = AxisRot3(Vector3(0, -1, 0), h*2*M_PI);

          session->endUpdate();
          Archon::Utilities::Thread::sleep(delay);
        }

        return 0;
      }
    }
  }
}

int main(int argc, const char *argv[]) throw()
{
  std::set_unexpected(Archon::X3D::Proxy::Exception::terminal);
  std::set_terminate(Archon::X3D::Proxy::Exception::terminal);

  return Archon::SaiTestApps::Clock::main(argc, argv);
}
```

Listing: **archon/sai_test_apps/cylinders/main.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#include <iostream>

#include <archon/math/vector.H>
#include <archon/util/time.H>
#include <archon/util/mutex.H>
#include <archon/util/ref.H>
#include <archon/util/options.H>

#include <archon/x3d/proxy/proxy.H>
```



```
namespace Archon
{
  namespace SaiTestApps
  {
    namespace Cylinders
    {
      using namespace std;
      using namespace Math;
      using namespace Utilities;
      using namespace X3D::Proxy;

      void handleCyl(Time, int i) { cout << "cyl" << i << "\n"; }

      int main(int argc, const char *argv[])
      {
        bool   opt_help              = false;
        int    opt_corbaEndpointPort = -1; // Auto assign
        string opt_serverName        = "archon";

        Options o;

        o.addSwitch("h", "help", opt_help, true,
                    "Describe the parameters");
        o.addConfig("p", "corba-endpoint-port", opt_corbaEndpointPort,
             opt_corbaEndpointPort,
                    "The TCP/IP port on which the CORBA Orb listens (-1 for
                        auto assign)",
                    Options::wantArg_always);
        o.addConfig("n", "server-name", opt_serverName, opt_serverName,
                    "The name of the server to connect to",
                    Options::wantArg_always);

        if(o.processCommandLine(argc, argv))
        {
          cerr << "Try --help\n";
          return 1;
        }

        if(opt_help)
        {
          cout <<
            "Test Application for 3D-Console by Brian Kristiansen & Kristian
                 Spangsege\n"
            "\n"
            "Synopsis: " << argv[0] << "\n"
            "\n"
            "Available options:\n";
          cout << o.list();
          return 0;
        }

        if(argc > 1)
        {
          cerr << "Too many aguments\n";
          cerr << "Try --help\n";
          return 1;
        }
```

```
Ref<Session> session =
  Session::create(opt_serverName, opt_corbaEndpointPort);

Ref<Application> application =
  Application::create(session, Uri());

Ref<Transform> rootTransform = Transform::create(application);
rootTransform->rotation = AxisRot3(Vector3(1, 0, 0), M_PI/2);
rootTransform->scale = Vector3(25, 25, 25);
rootTransform->translation = Vector3(0, 100, 0);

Ref<Group> rootGroup = Group::create(application);
rootGroup->children.add(rootTransform);

application->setRootGroup(rootGroup);
application->launch();

// 1
{
  Thread::sleep(Time(0.1));

  Ref<Appearance> apperance = Appearance::create(application);
  apperance->material = Material::create(application);

  Ref<Cylinder> cyl = Cylinder::create(application);
  cyl->side = false;
  cyl->top = false;
  cyl->bottom = false;

  Ref<Shape> shape = Shape::create(application);
  shape->appearance = apperance;
  shape->geometry = cyl;

  Ref<TouchSensor> touch = TouchSensor::create(application);
  touch->touchTime.registerInterest<SFTime>
    (new CookieFuncCaller<SFTime, int>(&handleCyl, 1));

  Ref<Transform> transform = Transform::create(application);
  transform->translation = Vector3(-5, 0, -1.5);
  transform->children.add(shape);
  transform->children.add(touch);

  rootTransform->children.add(transform);
}

// 2
{
  Thread::sleep(Time(0.1));

  Ref<Appearance> apperance = Appearance::create(application);
  apperance->material = Material::create(application);

  Ref<Cylinder> cyl = Cylinder::create(application);
  cyl->side = false;
  cyl->top = false;

  Ref<Shape> shape = Shape::create(application);
```



```
  shape->appearance = apperance;
  shape->geometry = cyl;

  Ref<TouchSensor> touch = TouchSensor::create(application);
  touch->touchTime.registerInterest<SFTime>
    (new CookieFuncCaller<SFTime, int>(&handleCyl, 2));

  Ref<Transform> transform = Transform::create(application);
  transform->translation = Vector3(-2, 0, -1.5);
  transform->children.add(shape);
  transform->children.add(touch);

  rootTransform->children.add(transform);
}

// 3
{
  Thread::sleep(Time(0.1));

  Ref<Appearance> apperance = Appearance::create(application);
  apperance->material = Material::create(application);

  Ref<Cylinder> cyl = Cylinder::create(application);
  cyl->side = false;
  cyl->bottom = false;

  Ref<Shape> shape = Shape::create(application);
  shape->appearance = apperance;
  shape->geometry = cyl;

  Ref<TouchSensor> touch = TouchSensor::create(application);
  touch->touchTime.registerInterest<SFTime>
    (new CookieFuncCaller<SFTime, int>(&handleCyl, 3));

  Ref<Transform> transform = Transform::create(application);
  transform->translation = Vector3(-5, 0, 1.5);
  transform->children.add(shape);
  transform->children.add(touch);

  rootTransform->children.add(transform);
}

// 4
{
  Thread::sleep(Time(0.1));

  Ref<Appearance> apperance = Appearance::create(application);
  apperance->material = Material::create(application);

  Ref<Cylinder> cyl = Cylinder::create(application);
  cyl->side = false;

  Ref<Shape> shape = Shape::create(application);
  shape->appearance = apperance;
  shape->geometry = cyl;

  Ref<TouchSensor> touch = TouchSensor::create(application);
  touch->touchTime.registerInterest<SFTime>
```

```
    (new CookieFuncCaller<SFTime, int>(&handleCyl, 4));

  Ref<Transform> transform = Transform::create(application);
  transform->translation = Vector3(-2, 0, 1.5);
  transform->children.add(shape);
  transform->children.add(touch);

  rootTransform->children.add(transform);
}

// 5
{
  Thread::sleep(Time(0.1));

  Ref<Appearance> apperance = Appearance::create(application);
  apperance->material = Material::create(application);

  Ref<Cylinder> cyl = Cylinder::create(application);
  cyl->top = false;
  cyl->bottom = false;

  Ref<Shape> shape = Shape::create(application);
  shape->appearance = apperance;
  shape->geometry = cyl;

  Ref<TouchSensor> touch = TouchSensor::create(application);
  touch->touchTime.registerInterest<SFTime>
    (new CookieFuncCaller<SFTime, int>(&handleCyl, 5));

  Ref<Transform> transform = Transform::create(application);
  transform->translation = Vector3(2, 0, -1.5);
  transform->children.add(shape);
  transform->children.add(touch);

  rootTransform->children.add(transform);
}

// 6
{
  Thread::sleep(Time(0.1));

  Ref<Appearance> apperance = Appearance::create(application);
  apperance->material = Material::create(application);

  Ref<Cylinder> cyl = Cylinder::create(application);
  cyl->top = false;

  Ref<Shape> shape = Shape::create(application);
  shape->appearance = apperance;
  shape->geometry = cyl;

  Ref<TouchSensor> touch = TouchSensor::create(application);
  touch->touchTime.registerInterest<SFTime>
    (new CookieFuncCaller<SFTime, int>(&handleCyl, 6));

  Ref<Transform> transform = Transform::create(application);
  transform->translation = Vector3(5, 0, -1.5);
  transform->children.add(shape);
```



```
  transform->children.add(touch);

  rootTransform->children.add(transform);
}

// 7
{
  Thread::sleep(Time(0.1));

  Ref<Appearance> apperance = Appearance::create(application);
  apperance->material = Material::create(application);

  Ref<Cylinder> cyl = Cylinder::create(application);
  cyl->bottom = false;

  Ref<Shape> shape = Shape::create(application);
  shape->appearance = apperance;
  shape->geometry = cyl;

  Ref<TouchSensor> touch = TouchSensor::create(application);
  touch->touchTime.registerInterest<SFTime>
    (new CookieFuncCaller<SFTime, int>(&handleCyl, 7));

  Ref<Transform> transform = Transform::create(application);
  transform->translation = Vector3(2, 0, 1.5);
  transform->children.add(shape);
  transform->children.add(touch);

  rootTransform->children.add(transform);
}

// 8
{
  Thread::sleep(Time(0.1));

  Ref<Appearance> apperance = Appearance::create(application);
  apperance->material = Material::create(application);

  Ref<Cylinder> cyl = Cylinder::create(application);

  Ref<Shape> shape = Shape::create(application);
  shape->appearance = apperance;
  shape->geometry = cyl;

  Ref<TouchSensor> touch = TouchSensor::create(application);
  touch->touchTime.registerInterest<SFTime>
    (new CookieFuncCaller<SFTime, int>(&handleCyl, 8));

  Ref<Transform> transform = Transform::create(application);
  transform->translation = Vector3(5, 0, 1.5);
  transform->children.add(shape);
  transform->children.add(touch);

  rootTransform->children.add(transform);
}

Archon::Utilities::Time delay(0, 1000000001);
```

```
        while(true)
        {
          Archon::Utilities::Thread::sleep(delay);
        }

        return 0;
      }
    }
  }
}

int main(int argc, const char *argv[]) throw()
{
  std::set_unexpected(Archon::X3D::Proxy::Exception::terminal);
  std::set_terminate(Archon::X3D::Proxy::Exception::terminal);

  return Archon::SaiTestApps::Cylinders::main(argc, argv);
}
```

Listing: **archon/sai_test_apps/filebrowser/main.C**

```
#include <sys/types.h>
//#include <sys/stat.h>
#include <dirent.h>
#include <unistd.h>
#include <fcntl.h>
#include <signal.h>
#include <stropts.h>

#include <iostream>

#include <archon/math/vector.H>
#include <archon/util/thread.H>
#include <archon/util/ref.H>
#include <archon/util/options.H>
#include <archon/util/file.H>
#include <archon/util/text.H>

#include <archon/x3d/proxy/proxy.H>

using namespace std;
using namespace Archon;
using namespace Archon::Math;
using namespace Archon::Utilities;
using namespace Archon::X3D::Proxy;

namespace Archon
{
  namespace SaiTestApps
  {
    namespace Filebrowser
    {
      Mutex mutex;
      Condition directoryRefresh(mutex);

      string path;

      Ref<Shape> inactiveBoxShape;
```



```
      Ref<Shape> activeBoxShape;

      void handleSignalIO(int s)
      {
        directoryRefresh.notifyAll();
      }

      void handleBack(Time)
      {
        {
          Mutex::Lock l(mutex);
          chdir("..");
          path = File::getCWD();
        }
        directoryRefresh.notifyAll();
      }

      void handleSubdir(Time, string dir)
      {
        {
          Mutex::Lock l(mutex);
          chdir(dir.c_str());
          path = File::getCWD();
        }
        directoryRefresh.notifyAll();
      }

      void handleText(Time, string file)
      {
        {
          cerr << "FILE=" << file << "\n";
          Mutex::Lock l(mutex);
          string s = "archon_textviewer " + file + " &";
          system(s.c_str());
        }
      }

      void handleX3D(Time, string file)
      {
        {
          cerr << "FILE=" << file << "\n";
          Mutex::Lock l(mutex);
          string s = "archon_x3d_viewer " + file + " &";
          system(s.c_str());
        }
      }

      void handlePDB(Time, string file)
      {
        {
          cerr << "FILE=" << file << "\n";
          Mutex::Lock l(mutex);
          string s = "archon_molecule " + file + " &";
          system(s.c_str());
        }
      }

      void handleExecutable(Time, string file)
```

```
{
  {
    cerr << "FILE=" << file << "\n";
    Mutex::Lock l(mutex);
    string s = file + " &";
    system(s.c_str());
  }
}

class TextBox: public virtual RefObjectBase
{
  string textString;
  Ref<Application> application;
  Ref<Transform> transform;
public:
  TextBox(string s, Ref<Application> a)
  {
    textString = s;
    application = a;

    // Box
    Ref<Transform> boxTransform = Transform::create(application);
    boxTransform->scale = Vector3(3, 0.8, 0.1);
    boxTransform->children.add(inactiveBoxShape);

    // Text
    Ref<Transform> textTransform = Transform::create(application);
    textTransform->translation = Vector3(-2.9, 0, 0.11);

    Ref<FontStyle> fontStyle = FontStyle::create(application);
    vector<string> justifyStrings;
    justifyStrings.push_back("BEGIN");
    justifyStrings.push_back("MIDDLE");
    fontStyle->justify = justifyStrings;

    Ref<X3D::Proxy::Text> text = X3D::Proxy::Text::create(application);
    text->string.add(textString);
    text->maxExtent = 5.8;
    text->fontStyle = fontStyle;

    Ref<Shape> textShape = Shape::create(application);
    Ref<Appearance> textApp= Appearance::create(application);
    Ref<Material> textMaterial = Material::create(application);
    textMaterial->diffuseColor = Vector3(0.7, 0.7, 0.2);
    textApp->material = textMaterial;
    textShape->appearance = textApp;
    textShape->geometry = text;
    textTransform->children.add(textShape);

    // Transform
    transform = Transform::create(application);
    transform->children.add(boxTransform);
    transform->children.add(textTransform);
  }

  Ref<Transform> getTransform()
  {
    return transform;
```



```
  }

};


class FileScan
{
private:
  DIR *fd;

public:
  FileScan(string s, void(*func)(int))
  {
    struct sigaction sa;
    sa.sa_handler = func;

    if(sigaction(SIGIO, &sa, 0))
    {
      cerr << "Signal handler NOT installed\n";
      abort();
    }

    fd = opendir(s.c_str());
    int fd_dir = dirfd(fd);
    if(fd_dir == -1)
    {
      cerr << "Directory -> FD failed\n";
      abort();
    }

    if(fcntl(fd_dir, F_SETFL, O_ASYNC) == -1)
    {
      cerr << "IOCTL NOT called succesfully\n";
      abort();
    }

    if(fcntl(fd_dir, F_NOTIFY, DN_MULTISHOT|DN_CREATE|DN_DELETE|
        DN_RENAME) == -1)
    {
      cerr << "IOCTL NOT called succesfully\n";
      abort();
    }
  }

  ~FileScan()
  {
    if(fd)
      if(closedir(fd))
      {
        cerr << "IO-operation (close) failed.\n";
        abort();
      }
  }
};

int fileSort(const void *a, const void *b)
{
  struct stat a_filestat;
```

```
  struct stat b_filestat;


  lstat((*(const struct dirent **)(a))->d_name, &a_filestat);
  lstat((*(const struct dirent **)(b))->d_name, &b_filestat);

  if(S_ISDIR(a_filestat.st_mode) && !S_ISDIR(b_filestat.st_mode))
    return -1;
  else if(!S_ISDIR(a_filestat.st_mode) && S_ISDIR(b_filestat.st_mode))
    return 1;
  else
    return alphasort(a, b);
}

void drawDirectory(string fullPath, Ref<Application> application, Ref<
     GroupingNode> rootGroup)
{
  struct stat filestat;
  struct dirent **namelist;
  int n;
  int radius=30;
  int x=0;
  int y=11;
  vector<Ref<ChildNode> > children0;
  Ref<Transform> rootTransform;

  // Setup Point light
  Ref<Transform> lightTransform = Transform::create(application);
  lightTransform->translation = Vector3(0, 10, 0);
  Ref<PointLight> light = PointLight::create(application);
  light->location = Vector3(0, 10, 0);
  light->on = true;
  light->color = Vector3(1, 1, 1);
  light->intensity = 0.7;
  lightTransform->children.add(light);
  children0.push_back(lightTransform);

  n = scandir(fullPath.c_str(), &namelist, 0, fileSort);
  if(n < 0)
  {
    cerr << "scandir error\n";
    abort();
  }
  else
  {
    for(int i=0;i<n;++i)
    {
      if(strcmp(namelist[i]->d_name,".") == 0)
      {
        free(namelist[i]);
        continue;
      }

      rootTransform = Transform::create(application);
      rootTransform->rotation = AxisRot3(Vector3(0, -1, 0), x*6.05/
           radius);
      rootTransform->translation = Vector3(0, 0, 0);
```



```
      if(strcmp(namelist[i]->d_name,"..") == 0)
      {

        Ref<TextBox> textBox = new TextBox("../", application);
        Ref<Transform> tr = textBox->getTransform();
        tr->translation = Vector3(0, y*1.8, -radius);

        Ref<TouchSensor> touch = TouchSensor::create(application);
        touch->touchTime.registerInterest<SFTime>(new FuncCaller<SFTime
             >(handleBack));
        rootTransform->children.add(tr);
        rootTransform->children.add(touch);
        children0.push_back(rootTransform);

        free(namelist[i]);
        y--;
        if(y<1)
        {
          y=11;
          x++;
        }
        continue;
      }

      string s;
      if(fullPath == "/")
        s = fullPath + static_cast<string>(namelist[i]->d_name);
      else
        s = fullPath + "/" + static_cast<string>(namelist[i]->d_name);
      lstat(s.c_str(), &filestat);
      int perm = access(s.c_str(), X_OK);

      s = static_cast<string>(namelist[i]->d_name);

      if(S_ISDIR(filestat.st_mode))
      {
        Ref<TextBox> textBox = new TextBox(s + "/", application);
        Ref<Transform> tr = textBox->getTransform();
        tr->setTranslation(Vector3(0, y*1.8, -radius));

        if(perm==0)
        {
          Ref<TouchSensor> touch = TouchSensor::create(application);
          touch->touchTime.registerInterest<SFTime>(new
               CookieFuncCaller<SFTime, string>(handleSubdir, s));
          rootTransform->children.add(touch);
        }
        rootTransform->children.add(tr);
      }
      else
      {
        Ref<TextBox> textBox = new TextBox(s, application);
        Ref<Transform> tr = textBox->getTransform();
        tr->translation = Vector3(0, y*1.8, -radius);

        if(perm==0)
        {
          Ref<TouchSensor> touch = TouchSensor::create(application);
```

```
          touch->touchTime.registerInterest<SFTime>(new
               CookieFuncCaller<SFTime, string>(handleExecutable, s));
          rootTransform->children.add(touch);
        }
        else if(Archon::Utilities::Text::isSuffixOf(".x3d", s, true))
        {
          Ref<TouchSensor> touch = TouchSensor::create(application);
          touch->touchTime.registerInterest<SFTime>(new
               CookieFuncCaller<SFTime, string>(handleX3D, s));
          rootTransform->children.add(touch);
        }
        else if(Archon::Utilities::Text::isSuffixOf(".pdb", s, true))
        {
          Ref<TouchSensor> touch = TouchSensor::create(application);
          touch->touchTime.registerInterest<SFTime>(new
               CookieFuncCaller<SFTime, string>(handlePDB, s));
          rootTransform->children.add(touch);
        }
        else
        {
          Ref<TouchSensor> touch = TouchSensor::create(application);
          touch->touchTime.registerInterest<SFTime>(new
               CookieFuncCaller<SFTime, string>(handleText, s));
          rootTransform->children.add(touch);
        }

        rootTransform->children.add(tr);
      }
      children0.push_back(rootTransform);
      free(namelist[i]);
      y--;
      if(y<1)
      {
        y=11;
        x++;
      }
    }
    free(namelist);
  }

  children0.push_back(rootTransform);
  rootGroup->children = children0;
}

int main(int argc, const char *argv[]) throw()
{
  bool   opt_help               = false;
  int    opt_corbaEndpointPort = -1; // Auto assign
  string opt_serverName         = "archon";

  Options o;

  o.addSwitch("h", "help", opt_help, true,
            "Describe the parameters");
  o.addConfig("p", "corba-endpoint-port", opt_corbaEndpointPort,
       opt_corbaEndpointPort,
            "The TCP/IP port on which the CORBA Orb listens (-1 for
                 auto assign)",
```



```
            Options::wantArg_always);
o.addConfig("n", "server-name", opt_serverName, opt_serverName,
            "The name of the server to connect to",
            Options::wantArg_always);

if(o.processCommandLine(argc, argv))
{
  cerr << "Try --help\n";
  return 1;
}

if(opt_help)
{
  cout <<
    "Test Application for 3D-Console by Brian Kristiansen & Kristian
         Spangsege\n"
    "\n"
    "Synopsis: " << argv[0] << "\n"
    "\n"
    "Available options:\n";
  cout << o.list();
  return 0;
}

if(argc > 2)
{
  cerr << "Too many aguments\n";
  cerr << "Try --help\n";
  return 1;
}

if(argc == 2)
{
  path = argv[1];
  chdir(argv[1]);
}
else path = File::getCWD();

Ref<Session> session =
  Session::create(opt_serverName, opt_corbaEndpointPort);

Ref<Application> application =
  Application::create(session, Uri());

Ref<Group> rootGroup = Group::create(application);

inactiveBoxShape = Shape::create(application);
Ref<Appearance> inactiveAppearance = Appearance::create(application);
Ref<Material> inactiveMaterial = Material::create(application);
inactiveMaterial->diffuseColor = Vector3(0.3, 0.3, 0.8);
inactiveMaterial->transparency = 0.7;
inactiveAppearance->material = inactiveMaterial;
Ref<Box> inactiveBox = Box::create(application);
inactiveBoxShape->geometry = inactiveBox;
inactiveBoxShape->appearance = inactiveAppearance;

activeBoxShape = Shape::create(application);
Ref<Appearance> activeAppearance = Appearance::create(application);
```

```
        Ref<Material> activeMaterial = Material::create(application);
        activeMaterial->diffuseColor = Vector3(0.5, 0.5, 0.8);
        activeMaterial->transparency = 0.2;
        activeAppearance->material = activeMaterial;
        Ref<Box> activeBox = Box::create(application);
        activeBoxShape->geometry = activeBox;
        activeBoxShape->appearance = activeAppearance;

        application->setRootGroup(rootGroup);
        application->launch();

        for(;;)
        {
          FileScan fs(path, handleSignalIO);
          session->beginUpdate();
          drawDirectory(path, application, rootGroup);
          session->endUpdate();
          directoryRefresh.wait();
        }

        return 0;
      }
    }
  }
}

int main(int argc, const char *argv[]) throw()
{
  std::set_unexpected(Archon::X3D::Proxy::Exception::terminal);
  std::set_terminate(Archon::X3D::Proxy::Exception::terminal);

  return Archon::SaiTestApps::Filebrowser::main(argc, argv);
}
```

Listing: **archon/sai_test_apps/simple/main.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#include <iostream>

#include <archon/math/vector.H>
```



```
#include <archon/util/time.H>
#include <archon/util/mutex.H>
#include <archon/util/ref.H>
#include <archon/util/options.H>

#include <archon/x3d/proxy/proxy.H>

namespace Archon
{
  namespace SaiTestApps
  {
    namespace Clock
    {
      using namespace std;
      using namespace Math;
      using namespace Utilities;
      using namespace X3D::Proxy;

      void isOver(bool v)
      {
        cerr << v << "\n";
      }

      int main(int argc, const char *argv[])
      {
        bool   opt_help              = false;
        int    opt_corbaEndpointPort = -1; // Auto assign
        string opt_serverName        = "archon";

        Options o;

        o.addSwitch("h", "help", opt_help, true,
                    "Describe the parameters");
        o.addConfig("p", "corba-endpoint-port", opt_corbaEndpointPort,
             opt_corbaEndpointPort,
                    "The TCP/IP port on which the CORBA Orb listens (-1 for
                         auto assign)",
                    Options::wantArg_always);
        o.addConfig("n", "server-name", opt_serverName, opt_serverName,
                    "The name of the server to connect to",
                    Options::wantArg_always);

        if(o.processCommandLine(argc, argv))
        {
          cerr << "Try --help\n";
          return 1;
        }

        if(opt_help)
        {
          cout <<
            "Test Application for 3D-Console by Brian Kristiansen & Kristian
                 Spangsege\n"
            "\n"
            "Synopsis: " << argv[0] << "\n"
            "\n"
            "Available options:\n";
          cout << o.list();
```

```
    return 0;
  }

  if(argc > 1)
  {
    cerr << "Too many aguments\n";
    cerr << "Try --help\n";
    return 1;
  }

  Ref<Session> session =
    Session::create(opt_serverName, opt_corbaEndpointPort);

  Ref<Application> application =
    Application::create(session, Uri());

  Ref<Sphere> sphere = Sphere::create(application);
  sphere->radius = 25;

  Ref<Material> material = Material::create(application);
  material->diffuseColor = Vector3(0.8, 0.3, 0.3);

  Ref<Appearance> apperance = Appearance::create(application);
  apperance->material = material;

  Ref<Shape> shape = Shape::create(application);
  shape->geometry = sphere;
  shape->appearance = apperance;

  Ref<Group> rootGroup = Group::create(application);
  rootGroup->children.add(shape);

  Ref<PointLight> light = PointLight::create(application);

  light->location = Vector3(0, 100, 0);

  rootGroup->children.add(light);

  Ref<TouchSensor> touch = TouchSensor::create(application);

  rootGroup->children.add(touch);

  touch->isOver.registerInterest<SFBool>(new FuncCaller<SFBool>(isOver
      ));
  //application->createRoute(touch->isOver, light->on);

  application->setRootGroup(rootGroup);
  application->launch();

  Archon::Utilities::Time delay(0, 100000000l);

  while(true)
  {
    Archon::Utilities::Thread::sleep(delay);
  }

  return 0;
}
```



```
    }
  }
}

int main(int argc, const char *argv[]) throw()
{
  std::set_unexpected(Archon::X3D::Proxy::Exception::terminal);
  std::set_terminate(Archon::X3D::Proxy::Exception::terminal);

  return Archon::SaiTestApps::Clock::main(argc, argv);
}
```

Listing: **archon/sai_test_apps/spot/main.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#include <iostream>

#include <archon/math/vector.H>
#include <archon/util/time.H>
#include <archon/util/thread.H>
#include <archon/util/ref.H>
#include <archon/util/options.H>
#include <archon/util/polygon_mesher.H>

#include <archon/x3d/proxy/proxy.H>

using namespace std;
using namespace Archon;
using namespace Archon::Math;
using namespace Archon::Utilities;
using namespace Archon::X3D::Proxy;

namespace Archon
{
  namespace SaiTestApps
  {
    namespace Spot
    {
      static bool quit = false;
```

```
Ref<SpotLight> redSpot;
Ref<SpotLight> greenSpot;
Ref<SpotLight> blueSpot;

void toggleRed(Time)
{
  static bool on = true;
  on = !on;
  if(redSpot) redSpot->on = on;
}

void toggleGreen(Time)
{
  static bool on = true;
  on = !on;
  if(greenSpot) greenSpot->on = on;
}

void toggleBlue(Time)
{
  static bool on = true;
  on = !on;
  if(blueSpot) blueSpot->on = on;
}

int main(int argc, const char *argv[])
{
  bool   opt_help              = false;
  int    opt_corbaEndpointPort = -1; // Auto assign
  string opt_serverName        = "archon";

  Options o;

  o.addSwitch("h", "help", opt_help, true,
              "Describe the parameters");
  o.addConfig("p", "corba-endpoint-port", opt_corbaEndpointPort,
       opt_corbaEndpointPort,
              "The TCP/IP port on which the CORBA Orb listens (-1 for
                   auto assign)",
              Options::wantArg_always);
  o.addConfig("n", "server-name", opt_serverName, opt_serverName,
              "The name of the server to connect to",
              Options::wantArg_always);

  if(o.processCommandLine(argc, argv))
  {
    cerr << "Try --help\n";
    return 1;
  }

  if(opt_help)
  {
    cout <<
      "Test Application for 3D-Console by Brian Kristiansen & Kristian
           Spangsege\n"
      "\n"
      "Synopsis: " << argv[0] << "\n"
      "\n"
```



```
      "Available options:\n";
    cout << o.list();
    return 0;
  }

  if(argc > 1)
  {
    cerr << "Too many aguments\n";
    cerr << "Try --help\n";
    return 1;
  }

  // Create a mesh
  double a = 10;
  vector<Vector3> outline;
  for(unsigned i=0; i<50; ++i)
  {
    double v = 2*M_PI/50*i;
    outline.push_back(Vector3(a*cos(v), 0, -a*sin(v)));
  }
  vector<Vector3> meshVertices;
  vector<int> meshPolygons;
  meshPolygon(outline, 0.25, meshVertices, meshPolygons);

  Ref<Session> session =
    Session::create(opt_serverName, opt_corbaEndpointPort);

  Ref<Application> application =
    Application::create(session, Uri());

  session->beginUpdate();

  Ref<Group> rootGroup = Group::create(application);

  Ref<Transform> rootTransform = Transform::create(application);
  rootTransform->scale = Vector3(10, 10, 10);
  rootTransform->translation = Vector3(0, 10, 0);
  rootGroup->children.add(rootTransform);

  Ref<Coordinate> coord = Coordinate::create(application);
  coord->point = meshVertices;
  Ref<IndexedFaceSet> indexedFaceSet = IndexedFaceSet::create(
       application);
  indexedFaceSet->coord = coord;
  indexedFaceSet->coordIndex = meshPolygons;
  Ref<Shape> shape = Shape::create(application);
  shape->geometry = indexedFaceSet;
  Ref<Appearance> appearance = Appearance::create(application);
  appearance->material = Material::create(application);

  /*
    Ref<ImageTexture> texture = ImageTexture::create(context);
    vector<string> url;
    url.push_back("clock.png");
    texture->setUrl(url);
    apperance->setTexture(texture);
  */
```

```
    shape->appearance = appearance;

    vector<Ref<ChildNode> > children1;
    children1.push_back(shape);

    vector<double> keys;
    keys.push_back(0.0/3);
    keys.push_back(1.0/3);
    keys.push_back(2.0/3);
    keys.push_back(3.0/3);

    vector<AxisRot3> keyValues;
    keyValues.push_back(AxisRot3(Vector3(cos(2*M_PI*0/3), 0, -sin(2*M_PI
         *0/3)), 0.3));
    keyValues.push_back(AxisRot3(Vector3(cos(2*M_PI*1/3), 0, -sin(2*M_PI
         *1/3)), 0.3));
    keyValues.push_back(AxisRot3(Vector3(cos(2*M_PI*2/3), 0, -sin(2*M_PI
         *2/3)), 0.3));
    keyValues.push_back(AxisRot3(Vector3(cos(2*M_PI*3/3), 0, -sin(2*M_PI
         *3/3)), 0.3));

    {
      Ref<SpotLight> spot = SpotLight::create(application);
      redSpot = spot;
      spot->color = Vector3(1, 0, 0);
      spot->beamWidth = M_PI*1/8;
      spot->cutOffAngle = M_PI*3/8;
      spot->direction = Vector3(0, -1, 0);
      Ref<Transform> transform = Transform::create(application);
      transform->translation = Vector3(2*cos(2*M_PI*0/3), 10, -2*sin(2*
           M_PI*0/3));
      Ref<OrientationInterpolator> orient = OrientationInterpolator::
           create(application);
      orient->key = keys;
      orient->keyValue = keyValues;
      application->createRoute(orient->value, transform->rotation);
      Ref<TimeSensor> time = TimeSensor::create(application);
      time->cycleInterval = Time(2.1);
      application->createRoute(time->fraction, orient->fraction);
      time->loop = true;


      Ref<Cylinder> cyl = Cylinder::create(application);
      cyl->radius = 0.2;
      cyl->height = 0.5;
      Ref<Material> mat = Material::create(application);
      mat->diffuseColor = Vector3(1, 0.5, 0.5);
      Ref<Appearance> appearance = Appearance::create(application);
      appearance->material = mat;
      Ref<Shape> shape = Shape::create(application);
      shape->geometry = cyl;
      shape->appearance = appearance;

      Ref<TouchSensor> touch = TouchSensor::create(application);
      touch->touchTime.registerInterest<SFTime>(new FuncCaller<SFTime>(
           toggleRed));

      vector<Ref<ChildNode> > children;
```



```
      children.push_back(spot);
      children.push_back(orient);
      children.push_back(time);
      children.push_back(shape);
      children.push_back(touch);
      transform->children = children;
      children1.push_back(transform);
    }

    {
      Ref<SpotLight> spot = SpotLight::create(application);
      greenSpot = spot;
      spot->color = Vector3(0, 1, 0);
      spot->beamWidth = M_PI*1/8;
      spot->cutOffAngle = M_PI*3/8;
      spot->direction = Vector3(0, -1, 0);
      Ref<Transform> transform = Transform::create(application);
      transform->translation = Vector3(2*cos(2*M_PI*1/3), 10, -2*sin(2*
           M_PI*1/3));
      Ref<OrientationInterpolator> orient = OrientationInterpolator::
           create(application);
      orient->key = keys;
      orient->keyValue = keyValues;
      application->createRoute(orient->value, transform->rotation);
      Ref<TimeSensor> time = TimeSensor::create(application);
      time->cycleInterval = Time(2.4);
      application->createRoute(time->fraction, orient->fraction);
      time->loop = true;

      Ref<Cylinder> cyl = Cylinder::create(application);
      cyl->radius = 0.2;
      cyl->height = 0.5;
      Ref<Material> mat = Material::create(application);
      mat->diffuseColor = Vector3(0.5, 1, 0.5);
      Ref<Appearance> appearance = Appearance::create(application);
      appearance->material = mat;
      Ref<Shape> shape = Shape::create(application);
      shape->geometry = cyl;
      shape->appearance = appearance;

      Ref<TouchSensor> touch = TouchSensor::create(application);
      touch->touchTime.registerInterest<SFTime>(new FuncCaller<SFTime>(
           toggleGreen));

      vector<Ref<ChildNode> > children;
      children.push_back(spot);
      children.push_back(orient);
      children.push_back(time);
      children.push_back(shape);
      children.push_back(touch);
      transform->children = children;
      children1.push_back(transform);
    }

    {
      Ref<SpotLight> spot = SpotLight::create(application);
      blueSpot = spot;
      spot->color = Vector3(0, 0, 1);
```

```
    spot->beamWidth = M_PI*1/8;
    spot->cutOffAngle = M_PI*3/8;
    spot->direction = Vector3(0, -1, 0);
    Ref<Transform> transform = Transform::create(application);
    transform->translation = Vector3(2*cos(2*M_PI*2/3), 10, -2*sin(2*
         M_PI*2/3));
    Ref<OrientationInterpolator> orient = OrientationInterpolator::
         create(application);
    orient->key = keys;
    orient->keyValue = keyValues;
    application->createRoute(orient->value, transform->rotation);
    Ref<TimeSensor> time = TimeSensor::create(application);
    time->cycleInterval = Time(2.7);
    application->createRoute(time->fraction, orient->fraction);
    time->loop = true;

    Ref<Cylinder> cyl = Cylinder::create(application);
    cyl->radius = 0.2;
    cyl->height = 0.5;
    Ref<Material> mat = Material::create(application);
    mat->diffuseColor = Vector3(0.5, 0.5, 1);
    Ref<Appearance> appearance = Appearance::create(application);
    appearance->material = mat;
    Ref<Shape> shape = Shape::create(application);
    shape->geometry = cyl;
    shape->appearance = appearance;

    Ref<TouchSensor> touch = TouchSensor::create(application);
    touch->touchTime.registerInterest<SFTime>(new FuncCaller<SFTime>(
         toggleBlue));

    vector<Ref<ChildNode> > children;
    children.push_back(spot);
    children.push_back(orient);
    children.push_back(time);
    children.push_back(shape);
    children.push_back(touch);
    transform->children = children;
    children1.push_back(transform);
  }

  rootTransform->children = children1;
  session->endUpdate();

  application->setRootGroup(rootGroup);
  application->launch();

  Archon::Utilities::Time delay(0, 100000000l);
  while(!quit)
  {
    session->beginUpdate();
    session->endUpdate();

    Archon::Utilities::Thread::sleep(delay);
  }

  return 0;
}
```



```
      }
    }
  }

int main(int argc, const char *argv[]) throw()
{
  std::set_unexpected(Archon::X3D::Proxy::Exception::terminal);
  std::set_terminate(Archon::X3D::Proxy::Exception::terminal);

  return Archon::SaiTestApps::Spot::main(argc, argv);
}
```

Listing: **archon/sai_test_apps/textviewer/main.C**

```
#include <sys/types.h>
#include <dirent.h>
#include <unistd.h>
#include <fcntl.h>
#include <signal.h>
#include <stropts.h>

#include <fstream>

#include <archon/math/vector.H>
#include <archon/util/thread.H>
#include <archon/util/ref.H>
#include <archon/util/options.H>
#include <archon/util/file.H>
#include <archon/util/polygon_mesher.H>

#include <archon/x3d/proxy/proxy.H>

using namespace std;
using namespace Archon;
using namespace Archon::Math;
using namespace Archon::Utilities;
using namespace Archon::X3D::Proxy;

namespace Archon
{
  namespace SaiTestApps
  {
    namespace Textviewer
    {
      Mutex mutex;
      Condition directoryRefresh(mutex);

      bool quit = false;

      void handleClose(Time)
      {
        quit = true;
        directoryRefresh.notifyAll();
      }

      void handleBack(Time)
      {
        directoryRefresh.notifyAll();
```

```
}

void handleNext(Time)
{
  directoryRefresh.notifyAll();
}

class TextPage: public virtual RefObjectBase
{
  string textString;
  Ref<Application> application;
  Ref<Transform> transform;
  Ref<X3D::Proxy::Text> text;
  Ref<Transform> nextButton;
public:
  TextPage(Ref<Application> a)
  {
    application = a;

    // Create a mesh
    vector<Vector3> outline;
    outline.push_back(Vector3(0, 0, 0));
    outline.push_back(Vector3(1, 0, 0));
    outline.push_back(Vector3(1, 1, 0));
    outline.push_back(Vector3(0, 1, 0));
    vector<Vector3> meshVertices;
    vector<int> meshPolygons;
    meshPolygon(outline, 0.05, meshVertices, meshPolygons);

    Vector3 paperSize(0.21, 0.297, 1);

    // Paper
    Ref<Transform> paperTransform = Transform::create(application);
    paperTransform->scale = paperSize;

    Ref<Coordinate> coord = Coordinate::create(application);
    coord->point = meshVertices;
    Ref<IndexedFaceSet> indexedFaceSet = IndexedFaceSet::create(
         application);
    indexedFaceSet->coord = coord;
    indexedFaceSet->coordIndex = meshPolygons;
    indexedFaceSet->solid = false;
    Ref<Shape> paperShape = Shape::create(application);
    Ref<Appearance> paperAppearance = Appearance::create(application);
    Ref<Material> paperMaterial = Material::create(application);
    paperMaterial->diffuseColor = Vector3(0.9, 0.9, 0.9);
    paperAppearance->material = paperMaterial;
    paperShape->geometry = indexedFaceSet;
    paperShape->appearance = paperAppearance;

    paperTransform->children.add(paperShape);

    // Text
    Ref<Transform> textTransform = Transform::create(application);
    textTransform->translation = Vector3(0.01, paperSize
         [1]-0.005, 0.001);
    textTransform->scale = paperSize/60.0;
```



```
    Ref<FontStyle> fontStyle = FontStyle::create(application);
    vector<string> familyStrings;
    familyStrings.push_back("TYPEWRITER");
    fontStyle->family = familyStrings;
    vector<string> justifyStrings;
    justifyStrings.push_back("BEGIN");
    justifyStrings.push_back("BEGIN");
    fontStyle->justify = justifyStrings;

    text = X3D::Proxy::Text::create(application);
    text->fontStyle = fontStyle;

    Ref<Shape> textShape = Shape::create(application);
    Ref<Appearance> textApp= Appearance::create(application);
    Ref<Material> textMaterial = Material::create(application);
    textMaterial->diffuseColor = Vector3(0, 0, 0);
    textApp->material = textMaterial;
    textShape->appearance = textApp;
    textShape->geometry = text;
    textTransform->children.add(textShape);

    // Buttons
    Ref<Transform> closeButton = Transform::create(application);
    closeButton->translation = Vector3(paperSize[0]-0.1, paperSize
         [1], 0);
    closeButton->scale = Vector3(0.01, 0.01, 0.01);
    Ref<Shape> b1Shape = Shape::create(application);
    Ref<Appearance> b1Appearance = Appearance::create(application);
    Ref<Material> b1Material = Material::create(application);
    Ref<Box> b1Geometry = Box::create(application);
    b1Shape->appearance = b1Appearance;
    b1Shape->geometry = b1Geometry;
    b1Appearance->material = b1Material;
    b1Material->diffuseColor = Vector3(1, 0, 0);
    b1Material->transparency = 0.4;
    Ref<TouchSensor> b1Touch = TouchSensor::create(application);
    b1Touch->touchTime.registerInterest<SFTime>(new FuncCaller<SFTime>(
         handleClose));
    closeButton->children.add(b1Shape);
    closeButton->children.add(b1Touch);

    nextButton = Transform::create(application);
    nextButton->translation = Vector3(paperSize[0], paperSize[1], 0);
    nextButton->scale = Vector3(0.01, 0.01, 0.01);
    Ref<Shape> b2Shape = Shape::create(application);
    Ref<Appearance> b2Appearance = Appearance::create(application);
    Ref<Material> b2Material = Material::create(application);
    Ref<Sphere> b2Geometry = Sphere::create(application);
    b2Shape->appearance = b2Appearance;
    b2Shape->geometry = b2Geometry;
    b2Appearance->material = b2Material;
    b2Material->diffuseColor = Vector3(0, 1, 0);
    b2Material->transparency = 0.4;
    Ref<TouchSensor> b2Touch = TouchSensor::create(application);
    b2Touch->touchTime.registerInterest<SFTime>(new FuncCaller<SFTime>(
         handleNext));
    nextButton->children.add(b2Shape);
    nextButton->children.add(b2Touch);
```

```
    // Transform
    transform = Transform::create(application);
    transform->children.add(paperTransform);
    transform->children.add(textTransform);
    transform->children.add(closeButton);
    transform->children.add(nextButton);
  }

  Ref<Transform> getTransform()
  {
    return transform;
  }

  Ref<X3D::Proxy::Text> getText()
  {
    return text;
  }

  Ref<Transform> getNextButton()
  {
    return nextButton;
  }

};


int main(int argc, const char *argv[])
{
  bool   opt_help              = false;
  int    opt_corbaEndpointPort = -1; // Auto assign
  string opt_serverName        = "archon";

  Options o;

  o.addSwitch("h", "help", opt_help, true,
              "Describe the parameters");
  o.addConfig("p", "corba-endpoint-port", opt_corbaEndpointPort,
      opt_corbaEndpointPort,
              "The TCP/IP port on which the CORBA Orb listens (-1 for
                  auto assign)",
              Options::wantArg_always);
  o.addConfig("n", "server-name", opt_serverName, opt_serverName,
              "The name of the server to connect to",
              Options::wantArg_always);

  if(o.processCommandLine(argc, argv))
  {
    cerr << "Try --help\n";
    return 1;
  }

  if(opt_help)
  {
    cout <<
      "Test Application for 3D-Console by Brian Kristiansen & Kristian
           Spangsege\n"
      "\n"
```



```
      "Synopsis: " << argv[0] << " FILE\n"
      "\n"
      "Available options:\n";
    cout << o.list();
    return 0;
  }

  // Treat 1st arg as URL of X3D file to load
  if(argc < 2)
  {
    cerr << "Too few arguments\n";
    cerr << "Try --help\n";
    return 1;
  }

  if(argc > 2)
  {
    cerr << "Too many aguments\n";
    cerr << "Try --help\n";
    return 1;
  }

  string name = argv[1];
  std::ifstream source(name.c_str());

  Mutex::Lock l(mutex);

  Ref<Session> session =
    Session::create(opt_serverName, opt_corbaEndpointPort);

  Ref<Application> application =
    Application::create(session, Uri());

  Ref<Group> rootGroup = Group::create(application);

  bool reachedEOF=false;
  string s;

  session->beginUpdate();
  Ref<TextPage> textPage = new TextPage(application);
  Ref<Transform> tr = textPage->getTransform();
  Ref<X3D::Proxy::Text> text = textPage->getText();
  tr->translation = Vector3(0, 30, 0);
  tr->scale = Vector3(100, 100, 100);
  rootGroup->children.add(tr);
  session->endUpdate();

  application->setRootGroup(rootGroup);
  application->launch();

  vector<string> textLines;

  for(;;)
  {
    //      session->beginUpdate();
    textLines.clear();
    for(int x=58; x>0; --x)
    {
```

```
          char ch;
          if(!source.get(ch))
          {
            reachedEOF = true;
            s = "";
          }
          else
          {
            source.putback(ch);
            getline(source, s);
            textLines.push_back(s);
          }
        }
        text->string = textLines;
        if(reachedEOF)
        {
          tr->children.remove(textPage->getNextButton());
        }
        //      session->endUpdate();
        directoryRefresh.wait();
        if(quit)
          return 0;
      }
      return 0;
    }
  }
  }
}

int main(int argc, const char *argv[]) throw()
{
  std::set_unexpected(Archon::X3D::Proxy::Exception::terminal);
  std::set_terminate(Archon::X3D::Proxy::Exception::terminal);

  return Archon::SaiTestApps::Textviewer::main(argc, argv);
}
```

Listing: **archon/sai_test_apps/x3d_viewer/main.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */
```



```
#include <iostream>

#include <archon/math/vector.H>
#include <archon/util/time.H>
#include <archon/util/mutex.H>
#include <archon/util/ref.H>
#include <archon/util/options.H>

#include <archon/x3d/proxy/proxy.H>

namespace Archon
{
  namespace SaiTestApps
  {
    namespace Clock
    {
      using namespace std;
      using namespace Math;
      using namespace Utilities;
      using namespace X3D::Proxy;

      int main(int argc, const char *argv[])
      {
        bool   opt_help              = false;
        int    opt_corbaEndpointPort = -1; // Auto assign
        string opt_serverName        = "archon";

        Options o;

        o.addSwitch("h", "help", opt_help, true,
                    "Describe the parameters");
        o.addConfig("p", "corba-endpoint-port", opt_corbaEndpointPort,
             opt_corbaEndpointPort,
                    "The TCP/IP port on which the CORBA Orb listens (-1 for
                         auto assign)",
                    Options::wantArg_always);
        o.addConfig("n", "server-name", opt_serverName, opt_serverName,
                    "The name of the server to connect to",
                    Options::wantArg_always);

        if(o.processCommandLine(argc, argv))
        {
          cerr << "Try --help\n";
          return 1;
        }

        Uri uri;
        string uriException;
        if(argc>1) try
        {
          uri = Uri(argv[1]);
        }
        catch(Uri::SyntaxException &e)
        {
          uriException = Uri::explain(e);
        }

        if(opt_help)
```

```
        {
          cout <<
            "Test Application for 3D-Console by Brian Kristiansen & Kristian
                 Spangsege\n"
            "\n"
            "Synopsis: " << argv[0] << " URI\n"
            "\n"
            "Available options:\n";
          cout << o.list();
          return 0;
        }

        // Treat 1st arg as URL of X3D file to load
        if(argc < 2)
        {
          cerr << "Too few arguments\n";
          cerr << "Try --help\n";
          return 1;
        }

        if(argc > 2)
        {
          cerr << "Too many aguments\n";
          cerr << "Try --help\n";
          return 1;
        }

        if(!uriException.empty())
        {
          cerr << uriException;
          exit(1);
        }

        Ref<Session> session =
          Session::create(opt_serverName, opt_corbaEndpointPort);

        Ref<Application> application = Application::load(session, uri);

        application->launch();

        Archon::Utilities::Time delay(0, 1000000001);

        while(true)
        {
          Archon::Utilities::Thread::sleep(delay);
        }

        return 0;
      }
    }
  }
}

int main(int argc, const char *argv[]) throw()
{
  std::set_unexpected(Archon::X3D::Proxy::Exception::terminal);
  std::set_terminate(Archon::X3D::Proxy::Exception::terminal);
```



```
  return Archon::SaiTestApps::Clock::main(argc, argv);
}
```

Listing: **archon/util/condition.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_UTILITIES_CONDITION_H
#define ARCHON_UTILITIES_CONDITION_H

#include <list>

#include <sys/select.h> // fd_set

#include <archon/util/exception.H>
#include <archon/util/time.H>
#include <archon/util/mutex.H>

namespace Archon
{
  namespace Utilities
  {
    using namespace std;

    struct ThreadTerminatedException: Utilities::StateException
    {
      ThreadTerminatedException(string l): Utilities::Exception(l) {}
    };

    /**
     * A condition queue for implementing thread synchronization.
     */
    struct Condition
    {
      struct Exception: virtual Utilities::Exception {};
      struct ResourceException: Exception, Utilities::ResourceException
      {
        ResourceException(string l, string m): Utilities::Exception(l, m) {}
      };
      struct StateException: Exception, Utilities::StateException
      {
```

```
    StateException(string l, string m): Utilities::Exception(l, m) {}
  };
  struct InternalException: Exception, Utilities::InternalException
  {
    InternalException(string l, string m): Utilities::Exception(l, m) {}
  };

  Condition(Mutex &m);
  ~Condition();

  void notifyOne()
  {
    const int e = pthread_cond_signal(&cond);
    if(e != 0)
      ARCHON_THROW1(InternalException,
                    "Attempt to signal on condition failed");
    Mutex::Lock l(sigThreadsMutex);
    if(sigThreads.size()) notifySigThreads();
  }

  void notifyAll()
  {
    const int e = pthread_cond_broadcast(&cond);
    if(e != 0)
      ARCHON_THROW1(InternalException,
                    "Attempt to broadcast on condition failed");
    Mutex::Lock l(sigThreadsMutex);
    if(sigThreads.size()) notifySigThreads();
  }

  /**
   * NOTE: The mutex associated with this condition MUST ALWAYS be
   * aquired by YOU when you call this method. DON'T FORGET!
   *
   * Note that there is generally no guarantee that the condition
   * waited for is fulfilled upon return from this
   * method. Therefore you must always re-check the condition
   * again and repat the wait until you discover that the
   * condition is fulfilled. Hopefully you knew this already.
   */
  void wait() throw(ThreadTerminatedException);

  /**
   * NOTE: The mutex associated with this condition MUST ALWAYS be
   * aquired by YOU when you call this method. DON'T FORGET!
   *
   * Like 'wait()' except that when the 'timeout' is reached the
   * waiting is aborted. Note the 'timeout' is an absolute time
   * value and not a delay.
   *
   * \return True if the timeout has been reached and false if the
   * condition was signalled. Also, false is returned when the
   * waiting is interrupted by a UNIX system signal.
   *
   * \sa Time
   */
  bool timedWait(Time timeout) throw(ThreadTerminatedException);
```



```
      /**
       * NOTE: The mutex associated with this condition MUST ALWAYS be
       * aquired by YOU when you call this method. DON'T FORGET!
       *
       * This method suspends the calling thread until one of the
       * following event occur: Any event that would resume
       * 'timedWait' on this condition and any event that would resume
       * the system call pselect(n, readfds, writefds, exceptfds, 0,
       * 0).
       *
       * The first four arguments behave exactly as for the POSIX
       * 1003.1-2001 systemcall 'pselect'.
       *
       * \return A positive value has the same meaning as the POSIX
       * 1003.1-2001 systemcall 'pselect'. Zero means that it times
       * out, and -1 means that it was awoken by a signal - proberbly
       * due to somebody that called this->notifyAll().
       *
       * \sa Time
       */
      int select(int n, fd_set *readfds, fd_set *writefds,
                 fd_set *exceptfds, Time timeout)
        throw(ThreadTerminatedException);

    private:
      friend struct Thread;

      pthread_cond_t cond;
      Mutex *mutex;

      Mutex sigThreadsMutex;
      list<pthread_t> sigThreads;

      Condition(const Condition &); // Hide
      Condition &operator=(const Condition &); // Hide

      void notifySigThreads();
    };
  }
}

#endif // ARCHON_UTILITIES_CONDITION_H
```

Listing: **archon/util/condition.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
```

```
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#include <errno.h>
#include <time.h>
#include <stdlib.h>
#include <signal.h>
#include <iostream>

#include <archon/util/condition.H>
#include <archon/util/thread.H>

namespace Archon
{
  namespace Utilities
  {
    using namespace std;

    Condition::Condition(Mutex &m): mutex(&m)
    {
      const int e = pthread_cond_init(&cond, 0);
      if(e != 0)
        ARCHON_THROW1(ResourceException,
                      "Could not initialize condition");
    }

    Condition::~Condition()
    {{ // The extra scope is needed to work around gcc3.2 bug #8287
      const int e = pthread_cond_destroy(&cond);
      if(e != 0)
      {
        if(e == EBUSY)
          ARCHON_THROW1(StateException,
                        "Attempt to destroy condition on which "
                        "threads are waiting");
        ARCHON_THROW1(InternalException,
                      "Attempt to destroy condition failed");
      }
    }}

    void Condition::wait() throw(ThreadTerminatedException)
    {
      Ref<Thread> self = Thread::self();
      {
        Mutex::Lock l(self->waitConditionMutex);
        self->waitCondition = this;
      }

      int error = 0;
      if(!self->terminateRequest)
        error = pthread_cond_wait(&cond, &mutex->mutex);

      {
        Mutex::Lock l(self->waitConditionMutex);
        self->waitCondition = 0;
```



```
      }

      if(error) ARCHON_THROW1(InternalException,
                              "pthread_cond_wait failed");
      if(self->terminateRequest) ARCHON_THROW(ThreadTerminatedException);
    }

    bool Condition::timedWait(Time timeout)
      throw(ThreadTerminatedException)
    {
      Ref<Thread> self = Thread::self();
      {
        Mutex::Lock l(self->waitConditionMutex);
        self->waitCondition = this;
      }

      int error = 0;
      if(!self->terminateRequest)
        error = pthread_cond_timedwait(&cond, &mutex->mutex, &timeout.ts);

      {
        Mutex::Lock l(self->waitConditionMutex);
        self->waitCondition = 0;
      }

      bool timedout = false;
      if(error)
      {
        if(error == ETIMEDOUT) timedout = true;
        else if(error != EINTR)
          ARCHON_THROW1(InternalException,
                        "pthread_cond_timedwait failed");
      }
      if(self->terminateRequest) ARCHON_THROW(ThreadTerminatedException);
      return timedout;
    }


    namespace
    {
      const int signo = SIGUSR1;
      bool setupSignalDone = false;
      void sigDummy(int) {}
      void setupSignal()
      {
        struct sigaction sa, saOriginal;
        sigaction(signo, 0, &sa);
        saOriginal = sa;
        sa.sa_handler = &sigDummy;
        sigemptyset(&sa.sa_mask);
        sa.sa_flags &= ~(SA_RESTART|SA_RESETHAND);
        sa.sa_flags |= SA_NODEFER;
        sigaction(signo, &sa, 0);
      }
    }


    /**
```

```
 * \todo On Linux (at the time of writing) the pselect system call
 * has a race condition which allows it to miss a signal and thus
 * sleep forever. This is because it is a "stupid" wrapper around
 * the old 'select' systemcall. See
 * glibc-2.3.1/sysdeps/generic/pselect.c/__pselect. Maybe we
 * should split the sleep up into smal bits to prevent eternal
 * sleep.
 */
int Condition::select(int n, fd_set *readfds, fd_set *writefds,
                      fd_set *exceptfds, Time timeout)
  throw(ThreadTerminatedException)
{
  Time now = Time::now();
  if(timeout < now) timeout = Time();
  else timeout -= now;

  Ref<Thread> self = Thread::self();
  {
    Mutex::Lock l(self->waitConditionMutex);
    self->waitCondition = this;
  }

  int m = 0;
  int e;
  if(!self->terminateRequest)
  {
    // Block USR1 signal
    sigset_t oldSigset;
    sigset_t blockSigset;
    sigemptyset(&blockSigset);
    sigaddset(&blockSigset, signo);
    if(pthread_sigmask(SIG_BLOCK, &blockSigset, &oldSigset)!=0)
      ARCHON_THROW1(InternalException,
                    "pthread_sigmask failed");

    if(!setupSignalDone)
    {
      setupSignalDone = true;
      setupSignal();
    }

    list<pthread_t>::iterator sigThread;
    {
      Mutex::Lock l(sigThreadsMutex);
      sigThreads.push_back(pthread_self());
      sigThread = --sigThreads.end();
    }

    mutex->unlock();

    sigset_t unblockSigset = oldSigset;
    sigdelset(&blockSigset, signo);
    m = pselect(n, readfds, writefds, exceptfds,
                &timeout.ts, &unblockSigset);
    e = errno;

    mutex->lock();
```



```
        {
          Mutex::Lock l(sigThreadsMutex);
          sigThreads.erase(sigThread);
        }

        // Restore signal mask
        if(pthread_sigmask(SIG_SETMASK, &oldSigset, 0)!=0)
          ARCHON_THROW1(InternalException,
                        "pthread_sigmask failed");
      }

      {
        Mutex::Lock l(self->waitConditionMutex);
        self->waitCondition = 0;
      }

      if(m < 0 && e != EINTR)
        ARCHON_THROW1(InternalException, "pselect failed: " +
                      string(strerror(e)));
      if(self->terminateRequest)
        ARCHON_THROW(ThreadTerminatedException);
      return m;
    }

    void Condition::notifySigThreads()
    {
      list<pthread_t>::iterator i = sigThreads.begin();
      while(i != sigThreads.end())
      {
        if(pthread_kill(*i, signo)!=0)
          ARCHON_THROW1(InternalException, "pthread_kill failed");
        ++i;
      }
    }
  }
}
```

Listing: **archon/util/mutex.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */
```

```
#ifndef ARCHON_UTILITIES_MUTEX_H
#define ARCHON_UTILITIES_MUTEX_H

#include <errno.h>
#include <pthread.h>

#include <archon/util/exception.H>

namespace Archon
{
  namespace Utilities
  {
    using namespace std;

    struct Thread;
    struct Condition;

    /**
     * A mutex for thread synchronization.
     */
    struct Mutex
    {
      struct Exception: virtual Utilities::Exception {};
      struct ResourceException: Exception, Utilities::ResourceException
      {
        ResourceException(string l, string m): Utilities::Exception(l, m) {}
      };
      struct StateException: Exception, Utilities::StateException
      {
        StateException(string l, string m): Utilities::Exception(l, m) {}
      };
      struct InternalException: Exception, Utilities::InternalException
      {
        InternalException(string l, string m): Utilities::Exception(l, m) {}
      };

      Mutex()
      {
        const int e = pthread_mutex_init(&mutex, 0);
        if(e != 0)
          ARCHON_THROW1(ResourceException,
                        "Mutex initialization failed");
      }

      ~Mutex()
      {{ // The extra scope is needed to work around gcc3.2 bug #8287
        const int e = pthread_mutex_destroy(&mutex);
        if(e != 0)
        {
          if(e == EBUSY)
            ARCHON_THROW1(StateException,
                          "Attempt to delete a locked mutex");
          ARCHON_THROW1(InternalException,
                        "Mutex destruction failed");
        }
      }}

      /**
```



```
       * A mutex lock holder.
       *
       * This class is used to lock and unlock a mutex.
       * In the simplest situation it is used like this:
       *
       * Mutex m;
       *
       * void f()
       * {
       *   ...
       *
       *   {
       *     Mutex::Lock l(m)
       *
       *     // Critical stuff
       *   }
       *
       *   ...
       * }
       *
       * That is, The costruction of Lock, when a mutex is passed as
       * argument, locks that mutex. Also, the implicit destruction at
       * the end of its scope unlocks the mutex.
       *
       * You may also construct a Lock without passing a mutex. This
       * feature together with the availability of the 'aquire' and
       * 'release' methods allows you to do locking that does not
       * strictly follow your scopes while still ensuring that the the
       * lock is release when you leave the scope of the Lock
       * variable eg. when a execption escapes that scope.
       *
       * Note: The individual Lock variable are not reentrant. That
       * is, only one thread must access a specific Lock variable at
       * any time (this includes the implicit destruction).
       *
       * \sa aquire
       * \sa release
       */
      struct Lock
      {
        /**
         * Create a mutex lock holder that does not hold a lock.
         * A lock may later me aquired with the 'aquire' method.
         *
         * \sa aquire
         */
        Lock(): mutex(0) {}

        /**
         * Create a mutex lock holder and aquire a lock on the mutex.
         */
        Lock(const Mutex &m): mutex(&m)
        {
          m.lock();
        }

        ~Lock()
        {{ // The extra scope is needed to work around gcc3.2 bug #8287
```

```
          release();
        }}

        /**
         * Aquire a lock on the specified mutex. If another lock was
         * already held it is first released.
         */
        void aquire(const Mutex &m)
        {
          release();
          m.lock();
          mutex = &m;
        }

        /**
         * Release the currently held mutex lock. If no lock was held,
         * nothing is done.
         */
        void release()
        {
          if(!mutex) return;
          mutex->unlock();
          mutex = 0;
        }

      private:
        friend struct Thread;

        const Mutex *mutex;

        Lock(const Lock &); // Hide
        Lock &operator=(const Lock &); // Hide
      };

    private:
      friend struct Condition;

      pthread_mutex_t mutex;

      Mutex(const Mutex &); // Hide
      Mutex &operator=(const Mutex &); // Hide

      /**
       * Note that the constness of the 'lock' and the 'unlock'
       * methods strictly speaking is nonsense, but it is faked to
       * allow you to implement methods that are both const and thread
       * safe without riquiring you to const_cast 'this' every time.
       */
      void lock() const
      {
        const int e =
          pthread_mutex_lock(const_cast<pthread_mutex_t *>(&mutex));
        if(e == 0) return;
        if(e == EDEADLK)
          ARCHON_THROW1(StateException,
                        "Attempt to lock a locked mutex");
        if(e == EINVAL)
          ARCHON_THROW1(InternalException,
```



```
                        "Attempt to lock uninitialized mutex");
        ARCHON_THROW1(InternalException,
                      "Attempt to lock mutex failed");
      }

      void unlock() const
      {
        const int e =
          pthread_mutex_unlock(const_cast<pthread_mutex_t *>(&mutex));
        if(e == 0) return;
        if(e == EPERM)
          ARCHON_THROW1(StateException,
                        "Attempt to unlock mutex that is locked by another
                             thread");
        if(e == EINVAL)
          ARCHON_THROW1(InternalException,
                        "Attempt to unlock uninitialized mutex");
        ARCHON_THROW1(InternalException,
                      "Attempt to unlock mutex failed");
      }
    };
  }
}

#endif // ARCHON_UTILITIES_MUTEX_H
```

Listing: **archon/util/ref.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_UTILITIES_REF_H
#define ARCHON_UTILITIES_REF_H

#include <string>
#include <vector>
#include <algorithm>

#include <archon/util/exception.H>
#include <archon/util/mutex.H>

#include <archon/util/cxx_demangle.H>
```

```
namespace Archon
{
  namespace Utilities
  {
    using namespace std;

    template<typename T> struct BackRef;
    struct RefMapBase;
    template<typename T> struct RefMap;
    struct Thread;

    struct RefException: virtual Utilities::Exception {};

    struct RefStateException: RefException, Utilities::StateException
    {
      RefStateException() {}
      RefStateException(string l, string m): Utilities::Exception(l, m) {}
    };

    struct ForwardDestroyedException: RefStateException
    {
      ForwardDestroyedException(string l): Utilities::Exception(l) {}
    };


    /**
     * \sa Ref(T *, RefNoLockTag)
     */
    struct RefNoLockTag {};

    /**
     *\sa Ref(T *, RefSafeIncTag)
     */
    struct RefSafeIncTag {};


    /**
     * Intrusive reference counted smart pointer.
     *
     * The intrusive smart pointer has one very nice feature compared
     * with non-intrusive alternatives, that is, it allows several
     * smart pointer to be constructed from the same regular pointer
     * without loosing track of the usage count. Also one can
     * seamlessly convert from smart pointer to regular pointer and
     * back again.
     *
     * The big downside is of cause that it can only be used to point
     * to objects that are derived from RefObjectBase which
     * contains the usage count.
     *
     * Note: Ref's may only be used to point to dynamically allocated
     * objects. Never to automatc objects on the stack.
     *
     * Constness:
     *
     * Although it is legal and convenient to work with Ref's to
     * "const" objects (ie. Ref<const MyType>), strickly seen, it is
```



```
     * nonsense. This is because strict constness would prevent us
     * from changing the usage count and consequently would not allow
     * us to create or destroy references to the object. Never the
     * less "fake constness" is usefull because it still allows us to
     * declare that the rest of the object may not be written to
     * through this samrt pointer, and as such the concept of logical
     * constness is preserved. Further more, the violations of the
     * constness is entirely encapsulated in the RefObjectBase and its
     * associated smart pointer. Finally, since instances of
     * RefObjectBase must always be allocated dynamically, physical
     * constness is never an issue.
     *
     * Re-entrance/thread safty:
     *
     * Reference counted smart pointer are extreemly useful in multi
     * threaded scenarios. That is, holding a smart pointer to some
     * object is a guarantee that the object is not deallocated yet.
     * This in turn guarantees that when some thread invokes a method
     * via the smart pointer, it is safe to the extent that the target
     * object vill not be deleted by some other thread while the
     * method is being executed.
     *
     * To be generally usefull it must be safe for multiple threads to
     * concurrently create and destroy references to the same
     * object. To accomodate this, any access to the usage count is
     * secured by a critical section which is enforced by a mutex
     * inside each RefObjectBase instance.
     *
     * \note Although the implicit manipulations of the usage count is
     * thread safe, access to the reference variable itself is not
     * thread safe. That is, the reference itself must not be accessed
     * by more that one thread at a time. It is the responsibility of
     * the application to ensure this.
     *
     * \sa RefObjectBase
     * \sa getSafeRef
     *
     * \note Do NOT create refrerence variables in a contructor to the
     * object being constructed. The immediate reason is that if an
     * exception is thrown within the constructor then the object will
     * be deallocated regardless of reference count and at some point
     * in time when your reference is destroyed it will also destroy
     * the object. It depends on the situation which one comes first,
     * but the second one will lead to disaster. Even if you could
     * guarantee than no exception is throw from the constructor,
     * there are still good reasons not to create refrerence variables
     * in a contructor to the object being constructed. One is that if
     * the reference gets destroyed during the execution of the
     * destructor such that the reference count drops back to zero
     * then the object will get deallocated even before the costructor
     * returns to the caller. The truth is that we cannot gain
     * controll over the destruction of an object until the costructor
     * returns.
     *
     * \note It is possible with a pre-declaration of a class A to
     * declare a member Ref<A> in a class B without a definition of A
     * in the compilation unit. It does however require that you
     * override the implicit constructor and destructor and define
```

```
 * them in an other compilation unit where A is defined. Also
 * every method whos signature involves Ref<A> must be used and
 * defined in a compilation unit where A is defined.
 */
template<typename T>
struct Ref
{
  typedef T ObjectType;

  Ref(): p(0) {}

  /**
   * A pointer to a _dynamically_ allocated instance of a class
   * inherited from RefObjectBase. Note "dynamically allocated"!
   */
  Ref(T *p): p(p) { inc(); }

  Ref(const Ref &r): p(r.p) { inc(); }
  template<typename U> Ref(const Ref<U> &r): p(r.p) { inc(); }

  // The extra scope is needed to work around gcc3.2 bug #8287
  ~Ref() {{ dec(); }}

  /**
   * With no argument this method breaks the reference to the
   * previously referenced object and makes this reference a null
   * reference. Otherwise it breaks the reference to the
   * previously referenced object and creates a new reference to
   * the object pointed to by the argument.
   */
  void reset(T *p=0);

  /**
   * Efficient swapping that avoids access to the referenced
   * object particularly its reference count.
   */
  void swap(Ref &r) { std::swap(p, r.p); }

  Ref &operator=(const Ref &);
  template<typename U> Ref &operator=(const Ref<U> &);

  template<typename U> bool operator==(const Ref<U> &) const;
  template<typename U> bool operator!=(const Ref<U> &) const;
  template<typename U> bool operator<(const Ref<U> &) const;
  template<typename U> bool operator>(const Ref<U> &) const;

  /**
   * \sa get
   */
  T *operator->() const;

  operator bool() const { return p; }

  /**
   * When using this method and 'operator->' in multi threaded
   * environments it is important to make sure that the returned
   * pointer remains valid. There are at least three ways to
   * ensure this: Applying the method to a constant Ref, Securing
```



```
   * all access to the Ref with a Mutex or applying the method to
   * a Ref on the stack of the executing thread.
   *
   * This is also the method you would use when you need to
   * dynamic cast.
   */
  T *get() const { return p; }

  /**
   * Is this the sole reference to some object?
   */
  bool sole() const { return p && p->refSole(); }


  /**
   * Can be used to create a reference in a context where the
   * object mutex is locked. It should only be used in refDispose.
   */
  Ref(T *p, RefNoLockTag): p(p) { incNoLock(); }


  /**
   * Contructs a real reference if the argument is not null and if
   * the use count of the referenced object is at least one,
   * otherwise it constructs a null reference.
   *
   * This constructor has no relevans in sigle threaded
   * applications since when the usage count drops to zero the
   * object is destroyed imediately. In multi threaded
   * environmets, the situation is different. Here one thread may
   * access a reference counted object in the period of time
   * between the point where the usage count drops to zero and the
   * point where it is completely destroyed. Note that this
   * normally implies that the accessing thread is using a
   * conventional pointer to access the object. It also implies
   * that another thread is discarding the last Ref reference to
   * the object.
   *
   * You may say "Why even bother with this situation? The thread
   * should be using the Ref references anyway, and then there
   * would be no need for this method.". Well, that is almost
   * true. Never the less, there are situations where this method
   * is helpfull, and where it makes sence to store a reference in
   * a convetional pointer.
   *
   * One such situation is when a thread (A) wishes to store a
   * reference to a reference counted object in the intent to
   * access it later. At the same time this thread does not want
   * to postpone the destruction of the object anymore that it
   * would have without a reference (the key issue). Now, this is
   * where the conventional pointer come into play. The only way
   * that a reference can be held without postponing the
   * destruction of the object, is by using a conventional pointer
   * instead of a Ref.
   *
   * The next problem is of course that the conventional pointer
   * may become dangeling (pointing to a destoyed object) at any
   * time because another thread (D) discards the last Ref
```

```
 * reference to the object. As such, the conventional pointer
 * will be of no use to tread A. The trick is to provide a way
 * for the object destructor to nullify the conventional pointer
 * held by thread A. Now, with appropriate use of mutexes thread
 * A can at any time determine if the object is still alive.
 *
 * Determining if the object is still alive is seldom
 * enough. Thread A wants to access the object. It wants to call
 * methods on it. This demand raises new concerns. First of all,
 * to be sure that the object stays alive during the access, we
 * must protect it with the same mutex (M) that also protects the
 * nullification performed by thread B. Now, lets say that the
 * object is of a class derived from the class with the
 * destructor that nullifies the poiner. This means that it is
 * possible that thread A accesses a partially destroyed
 * object. Therefor it is not safe to use virtual methods during
 * the access by thread A.
 *
 * Another problem with accessing the object from within mutex M
 * is that for the time the access takes, thread B will be
 * blocked trying to discard the last reference. This is of
 * cource not a problem if the access is fast.
 *
 * A third problem with accessing the object from within
 * mutex M is that thread A must be carefull not to attempt to
 * enter a mutex that is locked by thread B. This would cause a
 * deadlock. The thing that should always be avoided is for two
 * threads (A and B) to attempt to lock two mutexes in opposite
 * order. If more than two mutexes are involved the general rule
 * is that they must be aquired in the same order my all
 * threads.
 *
 * If we could raise the reference count of the object just
 * before the access and then lower it again afterwards, then it
 * seems that we would be able to solve all three problems
 * above. That is, the reference count will not drop to zero
 * during the access. Therefore the object cannot become
 * partially destroyed. Thread B will not be blocked since the
 * reference it discards is not the last one. And finally thread
 * A does not need to keep a lock on mutex M during the access,
 * and the risk of deadlock will then be smaller.
 *
 * So, it should not be too difficult to raise the reference
 * count of the object. Why not just construct a Ref from the
 * pointer while holding a lock on mutex M? Well, this will not
 * do, since if the reference has dropped to zero before the Ref
 * construction then the object will be deleted anyway when
 * thread A leave mutex M. Ouch!
 *
 * What we need is the 'Ref(T *, RefSafeIncTag)'
 * sonstructor. This one attempts to construct a new Ref
 * reference from a conventional pointer, but the news is that
 * if the usage count already has dropped to zero then it
 * creates a null reference. It does this atomically with
 * respect to the standard increment and decrement opreations of
 * the reference counted object.
 *
 * Ok, what must I do then?
```



```
   *
   * Do something like the following:
   *
   * struct A: Thread
   * {
   *   Mutex m;
   *   R *r;
   *
   *   void dissociate()
   *   {
   *     Mutex::Lock l(m);
   *     r = 0;
   *   }
   *
   *   void main()
   *   {
   *     Mutex::Lock l(m);
   *     Ref<R> safeRef(r, RefSafeIncTag());
   *     l.release();
   *     if(safeRef) safeRef->access();
   *   }
   * };
   *
   * struct R: RefObjectBase // Reference counted object
   * {
   *   A *a;
   *
   *   ~O
   *   {
   *     a->dissociate();
   *   }
   *
   *   void access()
   *   {
   *     ...
   *   }
   * };
   *
   * Finally note that to adhere to our previously stated demand
   * that thread A should not postpone the destruction of the
   * object, we should add a Thread::terminate(self) to the
   * 'dissociate' method above and make sure that whatever the
   * 'access' method does it will be willing to meet this
   * termination request.
   */
  Ref(T *p, RefSafeIncTag);

private:
  template<typename U> friend struct Ref;
  friend struct BackRef<T>;
  friend struct RefMap<T>;

  T *p;

  void inc() const { if(p) p->refInc(); }
  void dec() const { if(p) p->refDec(); }

  void incNoLock() const { if(p) p->refIncNoLock(); }
```

```
};



/**
 * Base class for smart pointer objects.
 *
 * Any class T whos objects are to be referenced by smart pointers
 * Ref<T> must derive from RefObjectBase. This is necessary
 * because it is RefObjectBase that contains the actual usage
 * count and the methods for manipulating it. The manipulation is
 * done automatically and behind the scenes by Ref<T>.
 *
 * Note: It is normally a good idea to derive virtually from
 * RefObjectBase as this allows you to use multiple inheritance
 * when you build up a hierachies of reference counted objects. An
 * example of this is the X3D node hierachy, A counter example is
 * the stream endpoint objects: ReaderBase and WriterBase. In this
 * case it is important that we get distinct copies of the
 * reference count when we combine them into a BasicPipe.
 *
 * Note also that, unless you overide the 'dispose' method to
 * something that never deletes, you must always ue dynamic
 * allocation for the classes derived from RefObjectBase. Never
 * use automatic objects on the stack and do not make them members
 * of other classes.
 *
 * \sa Ref
 * \sa RefObject
 */
struct RefObjectBase: protected virtual Mutex
{
  virtual ~RefObjectBase() {}

  unsigned long getUseCount() const;

protected:
  RefObjectBase(): useCount(0) {}

  /**
   * This method is called when the reference count drops to zero.
   * Normally this implies that the object must be deleted and
   * therefore this is alos the default action for this method.
   *
   * You may override this in you derived classes, and normally
   * you will want to do this if the time of destruction is
   * determined not only by the reference count. A good example of
   * this is the BasicPipe template which involves two distinct
   * reference counts. Only when both have reached zero the pipe
   * should be destroyed.
   *
   * This method is always called with a lock on the mutex. this
   * is to allow overriding methods to consistently check the
   * state of the object before determining whether it should be
   * deleted. However it is an error to delete the object while a
   * lock is held. Therefore the lock is passed as an
   * argument.
   *
```



```
   * \sa BasicPipe
   */
  virtual void refDispose(Mutex::Lock &);

  unsigned long getUseCountNoLock() const { return useCount; }

private:
  template<typename T> friend struct Ref;
  template<typename T> friend struct BackRef;
  friend struct BackRefObjectBase;
  friend struct RefMapBase;
  friend struct Thread;

  void refInc() const;
  void refDec() const;

  void refIncNoLock() const;

  /**
   * \sa Ref<T>::getSafeRef
   */
  virtual bool refTryInc() const;

  /**
   * Is this the object reference exactly once?
   */
  bool refSole() const;

  unsigned long useCount;
};


typedef Ref<RefObjectBase> RefAny;
typedef Ref<const RefObjectBase> RefAnyConst;


/**
 * Convenience class for adding a reference count to a type that
 * for some reason cannot or should not be made to derive from
 * RefObjectBase. Obvious examples are STL classes such as
 * "string" and "vector", or the fundamental integral types such
 * as "char" and "int".
 */
template<typename T>
struct RefObject: virtual RefObjectBase
{
  T value;
  RefObject(T v): value(v) {}
};



/**
 * BackRef is almost an identical copy of Ref, which allows you to
 * work with two sets of references to the same class. The default
 * meaning of this is that the referenced object gets deleted when
 * both kinds of references drops to zero. The primary motivation
```

```
 * for the BackRef is to allow two objects to refer to each other
 * (A refers to B and B refers BACK to A). Without the special
 * BackRef this would violate the acyclic clause of the Ref, and
 * lead directly to memory leak. Note however that, even with the
 * support of BackRef, working with cyclic references is a "black
 * art" with very many pitfalls for the uninitiated, especially
 * when the objects are accessed by multiple threads.
 *
 * To undestand the general idea of BackRef, assume that we have
 * an object A with a normal reference to an object B. Also, a
 * number of outside parties hold a normal reference to A. In this
 * case we expect A and consequently B to be deleted when the last
 * outside party drops its reference. Now, if we add to B a normal
 * reference to A, then we are in trouble, because A and B will
 * keep each other alive forever. So how do we "repair" this
 * situation. The trick is to distinguish between the references
 * to A held by the outside parties and the one held by B, thus
 * enabling us to detect when either one of them drops to zero,
 * and not just when both of them does. The question is then, what
 * shall we do when this happens, and that depends on the
 * situation, but normally we should immitate the job carried out
 * during object destruction. More specifically we should release
 * all forward references held locally. In the example at hand, we
 * could think of the references held by the outside parties and
 * the one held by A as "forward" references, and of the reference
 * to A held by B as a backward reference. Then when the forward
 * reference count drops to zero in A we could drop its reference
 * to B effectively leading to the deletion of both B and A.
 *
 * This idea of distinguishing between forward and backward
 * references scales nicely to larger hierachical object
 * structures and can be applied recursivly. The first challenge
 * you face is thus to mentally arrange your object structure in
 * such a way that the direction become obvious, and such that the
 * forward references and backward references satisfies the
 * acyclic clause individually.
 *
 * If you can guarantee that the forward references and backward
 * references within a specific collection of objects satisfy the
 * acyclic clause individually, and that when the forward
 * reference count for an arbitrary object A in the collection
 * drops to zero then every forward reference held by that object
 * to other object in the collection will be released and that
 * after this point then no new forward reference can be held by
 * object A to an object in the collection, then you also have a
 * guarantee that every object in the collection is deleted when
 * the last reference from an outside party is released. This is obvious
    since ... (what about re-rising up from zero?)
 */
template<typename T>
struct BackRef
{
  typedef T ObjectType;

  BackRef(): p(0) {}

  BackRef(const Ref<T> &r): p(r.get()) { inc(); }
  BackRef(const BackRef &r): p(r.p) { inc(); }
```



```
  template<typename U> BackRef(const BackRef<U> &r): p(r.p) { inc(); }

  // The extra scope is needed to work around gcc3.2 bug #8287
  ~BackRef() {{ dec(); }}

  /**
   * Break the reference to the previously referenced object and
   * makes this reference a null reference.
   */
  void reset() { reset(0); }

  /**
   * Break the reference to the previously referenced object and
   * creates a new reference to the specified object.
   */
  void reset(const Ref<T> &r) { reset(r.p); }

  /**
   * Efficient swapping that avoids access to the referenced
   * object particularly its reference count.
   */
  void swap(BackRef &r) { std::swap(p, r.p); }

  BackRef &operator=(const BackRef &);
  template<typename U> BackRef &operator=(const BackRef<U> &);

  template<typename U> bool operator==(const BackRef<U> &) const;
  template<typename U> bool operator!=(const BackRef<U> &) const;

  /**
   * \sa get
   */
  Ref<T> operator->() const throw(ForwardDestroyedException);

  operator bool() const { return p; }

  Ref<T> getRef() const throw(ForwardDestroyedException);

private:
  template<typename U> friend struct BackRef;

  T *p;

  void reset(T *p);

  void inc() const { if(p) p->backRefInc(); }
  void dec() const { if(p) p->backRefDec(); }
};


/**
 * Base class for objects supporting both forward and backward
 * references to it.
 *
 * \sa BackRef
 * \sa RefObjectBase
 */
struct BackRefObjectBase: virtual RefObjectBase
```

```
{
  unsigned long getBackUseCount() const;

protected:
  BackRefObjectBase(): backUseCount(0), forwardDestroyed(false) {}

  virtual void refDispose(Mutex::Lock &);

  /**
   * This method is called just before the forward reference count
   * drops to zero. The purpose of this method is to take
   * appropriate measures to ensure that all back references to
   * this object held by other objects and or threads will
   * eventually be destroyed. This normally involves towo things:
   * Clearing all forward references held by this object to
   * objects that directly or indirectly hold back references to
   * this object, ant to send asynchronious termination requests
   * to all threads that hold back references to this object
   * (except if the thread is known to complete relativly fast
   * without being terminated).
   *
   * It is your responsibility to implement this method in such a
   * way that it does not lead to creation of new forward
   * references to this object.
   *
   * You may safely assume that no other than the calling thread
   * will access the object from the point in time where this
   * method is called.
   */
  virtual void refForwardDestroy() = 0;

  unsigned long getBackUseCountNoLock() const { return backUseCount; }

private:
  template<typename T> friend struct Ref;
  template<typename T> friend struct BackRef;

  void backRefInc() const;
  void backRefDec() const;

  virtual bool refTryInc() const;

  unsigned long backUseCount;
  bool forwardDestroyed;
};




/**
 * Base class for all RefMaps.
 *
 * \sa RefMap<T>
 */
struct RefMapBase
{
protected:
```



```
  RefMapBase();
  ~RefMapBase();

  unsigned long size();

  string disp();

  unsigned long get(RefObjectBase *, bool create);

  RefAny operator()(unsigned long key) const
  {
    Mutex::Lock l(refMapMutex);
    return key < refs.size() ? refs[key].first : 0;
  }

  void remove(unsigned long key, unsigned long transCount)
  {
    if(!key) return;
    Mutex::Lock l(refMapMutex);
    if(key >= refs.size()) return;
    removeNoLock(refs[key].first, transCount);
  }

  void remove(RefObjectBase *r)
  {
    Mutex::Lock l(refMapMutex);
    removeNoLock(r, 0);
  }

  void clearNoLock(bool destroyRefs);

  Mutex refMapMutex;
  // Pairs of refs and translation count
  vector<pair<RefObjectBase *, unsigned long> > refs;

private:
  static const unsigned long hashSize = 127;

  vector<unsigned long> *chunks[hashSize];
  vector<unsigned long> unused;

  void removeNoLock(RefObjectBase *, unsigned long transCount);
};

/**
 * Map references on the server uniquely to references on the
 * client.
 *
 * This map offers very fast (constant time) translation of keys
 * to refs, and reasonably fast translation from refs to keys
 * (that is faster than sequential search). Further more it offers
 * strict determination of wether a key is in the map as a side
 * effect of the translation from key to ref. It also has the
 * property of using small intergers as keys. That is, new keys
 * will always be smaller that the currently largest one if such a
 * key is free. Otherwise it will be one plus the currently
 * largest one.
 */
```

```
template<typename T>
struct RefMap: private RefMapBase
{
  /**
   * Return the number of mapped references.
   */
  RefMapBase::size;

  /**
   * Display the hashing statistics of this map.
   */
  RefMapBase::disp;

  /**
   * If the specified reference is found in the map then its
   * corresponding key is returned, If the reference is not found
   * and 'create' is false then the null key is returned,
   * otherwise an association for the reference is added to the
   * map and the new unique key is returned.  As a special case,
   * if the null reference is passed, then the null key is always
   * returned.
   */
  unsigned long get(Ref<T> r, bool create=true)
  {
    return RefMapBase::get(const_cast<RefObjectBase *>
                           (static_cast<const RefObjectBase *>
                            (r.get())), create);
  }

  /**
   * Returns the reference associated with the specified key or
   * the null if the key is null or there is no reference
   * associated with the key.
   */
  Ref<T> operator()(unsigned long key) const
  {
    RefAny r = RefMapBase::operator()(key);
    return dynamic_cast<T *>(r.get());
  }

  /**
   * Removes the specified key and its reference association from
   * the map.
   *
   * \param transCount Number of translations from ref to key to
   * remove. Zero indicates 'all'.
   */
  void remove(unsigned long key, unsigned long transCount=0)
  {
    RefMapBase::remove(key, transCount);
  }

  /**
   * Removes the specified reference and its key association from
   * the map.
   */
  void remove(Ref<T> r)
  {
```



```
    RefMapBase::remove(const_cast<RefObjectBase *>
                      (static_cast<const RefObjectBase *>(r.get())));
  }

  void clear(vector<Ref<T> > *remains = 0)
  {
    if(remains)
    {
      remains->clear();
      remains->resize(size());
      Mutex::Lock l(refMapMutex);
      unsigned j = 0;
      for(unsigned i = 1; i<refs.size(); ++i)
      {
        RefObjectBase *r = refs[i].first;
        if(r) remains->operator[](j++).p = dynamic_cast<T *>(r);
      }
      RefMapBase::clearNoLock(false);
    }
    else
    {
      Mutex::Lock l(refMapMutex);
      RefMapBase::clearNoLock(true);
    }
  }
};
```

```
template<typename T>
inline void Ref<T>::reset(T *_p)
{
  if(p == _p) return;
  dec();
  p = _p;
  inc();
}

template<typename T>
inline Ref<T> &Ref<T>::operator=(const Ref &r)
{
  reset(r.p);
  return *this;
}

template<typename T> template<typename U>
inline Ref<T> &Ref<T>::operator=(const Ref<U> &r)
{
  reset(r.p);
  return *this;
}

template<typename T> template<typename U>
inline bool Ref<T>::operator==(const Ref<U> &r) const
{
  return p == r.p;
```

```
}

template<typename T> template<typename U>
inline bool Ref<T>::operator!=(const Ref<U> &r) const
{
  return p != r.p;
}

template<typename T> template<typename U>
inline bool Ref<T>::operator<(const Ref<U> &r) const
{
  return p < r.p;
}

template<typename T> template<typename U>
inline bool Ref<T>::operator>(const Ref<U> &r) const
{
  return p > r.p;
}

template<typename T>
inline T *Ref<T>::operator->() const
{
  if(p) return p;
  ARCHON_THROW1(RefStateException, "Dereferencing null");
}

template<typename T>
inline Ref<T>::Ref(T *_p, RefSafeIncTag): p(0)
{
  if(_p && _p->refTryInc()) p = _p;
}



inline void RefObjectBase::refInc() const
{
  Mutex::Lock l(*this);
  ++const_cast<RefObjectBase *>(this)->useCount;
}

inline void RefObjectBase::refDec() const
{
  Mutex::Lock l(*this);
  if(!--const_cast<RefObjectBase *>(this)->useCount)
    const_cast<RefObjectBase *>(this)->refDispose(l);
}

inline void RefObjectBase::refIncNoLock() const
{
  ++const_cast<RefObjectBase *>(this)->useCount;
}

inline void RefObjectBase::refDispose(Mutex::Lock &l)
{
  l.release();
  // Note: The validity of this self destruction relies on the
  // fact that no explicit or implict references to "this" are
```



```
  // made after the delete statement (keep it like that).
  delete this;
}

inline bool RefObjectBase::refTryInc() const
{
  Mutex::Lock l(*this);
  if(useCount)
  {
    ++const_cast<RefObjectBase *>(this)->useCount;
    return true;
  }
  return false;
}

inline bool RefObjectBase::refSole() const
{
  Mutex::Lock l(*this);
  return useCount == 1;
}

inline unsigned long RefObjectBase::getUseCount() const
{
  Mutex::Lock l(*this);
  return useCount;
}



template<typename T>
inline void BackRef<T>::reset(T *_p)
{
  if(p == _p) return;
  dec();
  p = _p;
  inc();
}

template<typename T>
inline BackRef<T> &BackRef<T>::operator=(const BackRef &r)
{
  reset(r.p);
  return *this;
}

template<typename T> template<typename U>
inline BackRef<T> &BackRef<T>::operator=(const BackRef<U> &r)
{
  reset(r.p);
  return *this;
}

template<typename T> template<typename U>
inline bool BackRef<T>::operator==(const BackRef<U> &r) const
{
  return p == r.p;
}
```

```
template<typename T> template<typename U>
inline bool BackRef<T>::operator!=(const BackRef<U> &r) const
{
  return p != r.p;
}

template<typename T>
inline Ref<T> BackRef<T>::getRef() const throw(ForwardDestroyedException)
{
  if(!p) return 0;
  Ref<T> r(p, RefSafeIncTag());
  if(!r) ARCHON_THROW(ForwardDestroyedException);
  return r;
}

template<typename T>
inline Ref<T> BackRef<T>::operator->() const
  throw(ForwardDestroyedException)
{
  if(!p) return 0;
  Ref<T> r(p, RefSafeIncTag());
  if(!r) ARCHON_THROW(ForwardDestroyedException);
  return r;
}



inline void BackRefObjectBase::backRefInc() const
{
  Mutex::Lock l(*this);
  ++const_cast<BackRefObjectBase *>(this)->backUseCount;
}

inline void BackRefObjectBase::backRefDec() const
{
  Mutex::Lock l(*this);
  if(!--const_cast<BackRefObjectBase *>(this)->backUseCount)
    const_cast<BackRefObjectBase *>(this)->refDispose(l);
}


inline bool BackRefObjectBase::refTryInc() const
{
  Mutex::Lock l(*this);
  if(useCount&&!forwardDestroyed)
  {
    ++const_cast<BackRefObjectBase *>(this)->useCount;
    return true;
  }
  return false;
}


inline void BackRefObjectBase::refDispose(Mutex::Lock &l)
{
  if(!useCount)
```



```
      {
        if(!backUseCount)
        {
          l.release();

          // Note: The validity of this self destruction relies on the
          // fact that no explicit or implict references to "this" are
          // made after the delete statement (keep it like that).
          delete this;
          return;
        }

        if(forwardDestroyed) return;
        forwardDestroyed = true;
        Ref<BackRefObjectBase> r(this, RefNoLockTag());
        l.release();
        r->refForwardDestroy();
      }
    }

    inline unsigned long BackRefObjectBase::getBackUseCount() const
    {
      Mutex::Lock l(*this);
      return backUseCount;
    }
  }
}

namespace std
{
  template<typename T>
  void swap(Archon::Utilities::Ref<T> &r,
            Archon::Utilities::Ref<T> &s)
  {
    r.swap(s);
  }

  template<typename T>
  void swap(Archon::Utilities::BackRef<T> &r,
            Archon::Utilities::BackRef<T> &s)
  {
    r.swap(s);
  }
}

#endif // ARCHON_UTILITIES_REF_H
```

Listing: **archon/util/ref.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
```

```
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#include <archon/util/text.H>
#include <archon/util/ref.H>

namespace Archon
{
  namespace Utilities
  {
    RefMapBase::RefMapBase()
    {
      for(unsigned i=0; i<hashSize; ++i) chunks[i] = 0;

      // Add null as the first entry to correspond with the null
      // index.
      refs.push_back(pair<RefObjectBase *, unsigned long>(0, 0));
    }

    RefMapBase::~RefMapBase()
    {
      for(unsigned i=0; i<hashSize; ++i) if(chunks[i]) delete chunks[i];
      for(unsigned i=1; i<refs.size(); ++i)
      {
        RefObjectBase *r = refs[i].first;
        if(r) r->refDec();
      }
    }

    unsigned long RefMapBase::size()
    {
      Mutex::Lock l(refMapMutex);
      return refs.size() - unused.size() - 1;
    }

    string RefMapBase::disp()
    {
      Mutex::Lock l(refMapMutex);
      string s;
      unsigned long m = 1;
      for(unsigned i=0; i<hashSize; ++i)
        if(chunks[i] && chunks[i]->size() > m) m = chunks[i]->size();
      char buffer[1024];
      for(unsigned i=0; i<hashSize; ++i)
      {
        unsigned long n = chunks[i] ? chunks[i]->size() : 0;
        string bar = string(unsigned(70UL*n/m), '+');
        sprintf(buffer, "%7lu |%-70s|\n", n, bar.c_str());
        s += string(buffer);
      }
      return s;
```



```
    }

    unsigned long RefMapBase::get(RefObjectBase *r, bool create)
    {
      if(!r) return 0;
      unsigned long h = reinterpret_cast<unsigned long>(r) % hashSize;
      vector<unsigned long>::iterator j;
      bool f = false;
      unsigned long index;
      Mutex::Lock l(refMapMutex);
      vector<unsigned long> *c = chunks[h];
      if(c)
      {
        vector<unsigned long>::iterator i = c->begin();
        while(i != c->end())
        {
          index = *i;
          if(index)
          {
            pair<RefObjectBase *, unsigned long> &p = refs[index];
            if(p.first == r)
            {
              ++p.second;
              return index;
            }
          }
          else if(!f)
          {
            f = true;
            j = i;
          }
          ++i;
        }
      }
      if(!create) return 0;

      if(unused.size())
      {
        index = unused.back();
        unused.pop_back();
        pair<RefObjectBase *, unsigned long> &p = refs[index];
        p.first = r;
        p.second = 1;
      }
      else
      {
        index = refs.size();
        refs.push_back(pair<RefObjectBase *, unsigned long>(r, 1));
      }

      r->refInc();

      if(f) *j = index;
      else
      {
        if(!c) c = chunks[h] = new vector<unsigned long>;
        c->push_back(index);
      }
```

```
    return index;
  }

  void RefMapBase::removeNoLock(RefObjectBase *r, unsigned long transCount)
  {
    if(!r) return;
    unsigned long h = reinterpret_cast<unsigned long>(r) % hashSize;
    vector<unsigned long> *c = chunks[h];
    if(!c) return;
    vector<unsigned long>::iterator i = c->begin();
    while(i != c->end())
    {
      unsigned long index = *i;
      pair<RefObjectBase *, unsigned long> &p = refs[index];
      if(p.first == r)
      {
        if(transCount)
        {
          if(transCount > p.second) transCount = p.second;
          p.second -= transCount;
          if(p.second) return;
        }
        *i = 0;
        r->refDec();
        p.first = 0;
        p.second = 0;
        unused.push_back(index);
        return;
      }
      ++i;
    }
  }

  void RefMapBase::clearNoLock(bool destroyRefs)
  {
    if(destroyRefs)
      for(unsigned i=1; i<refs.size(); ++i)
      {
        RefObjectBase *r = refs[i].first;
        if(r) r->refDec();
      }
    for(unsigned i=0; i<hashSize; ++i)
    {
      if(!chunks[i]) continue;
      delete chunks[i];
      chunks[i] = 0;
    }
    refs.clear();
    unused.clear();
    refs.push_back(pair<RefObjectBase *, unsigned long>(0, 0));
  }
 }
}
```

Listing: **archon/util/thread.H**

```
/*
```



```
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_UTILITIES_THREAD_H
#define ARCHON_UTILITIES_THREAD_H

#include <list>

#include <sys/select.h> // fd_set

#include <archon/util/exception.H>
#include <archon/util/time.H>
#include <archon/util/condition.H>
#include <archon/util/ref.H>

namespace Archon
{
  namespace Utilities
  {
    using namespace std;

    /**
     * A thread of execution (aka light weight proccess or LWP).
     *
     * This class supports safe cancelation/termination of
     * threads. The problem of thread cancelation usually is that it
     * is hard to guarantee proper cleanup of allocated memory and
     * other resources. This is especially true in C++ programs where
     * it is crucial that the stack is noramlly unwinded ensuring that
     * class detructors gets called. So what we really need is to
     * allow one thread to provoke an exception in another
     * thread. This is exactly what this thread class supports. If a
     * thread gets a terminate request, then any current or future
     * calls to any of the following methods will raise a
     * TerminatedException.
     *
     * Semaphore::down
     * Condition::wait,
     * Condition::timedWait,
     * Condition::select,
     * Thread::wait,
     * Thread::sleep,
     * Thread::acceptTermination,
```

```
   *
   * \todo Implement thread priority.
   */
  struct Thread: virtual RefObjectBase
  {
    struct Exception: virtual Utilities::Exception {};
    struct ArgumentException: Exception, Utilities::ArgumentException
    {
      ArgumentException(string l, string m): Utilities::Exception(l, m) {}
    };
    struct ResourceException: Exception, Utilities::ResourceException
    {
      ResourceException(string l, string m): Utilities::Exception(l, m) {}
    };
    struct StateException: Exception, Utilities::StateException
    {
      StateException() {}
      StateException(string l, string m): Utilities::Exception(l, m) {}
    };
    struct InternalException: Exception, Utilities::InternalException
    {
      InternalException(string l, string m): Utilities::Exception(l, m) {}
    };
    struct AlreadyStartedException: StateException
    {
      AlreadyStartedException(string l): Utilities::Exception(l) {}
    };
    struct NotStartedException: StateException
    {
      NotStartedException(string l): Utilities::Exception(l) {}
    };


    /**
     * Let a new thread invoke a method of your choise on a
     * reference counted object of your choise. An extra reference
     * count is held during the execution of the specified method,
     * and it is released afterwards.
     *
     * \note This method is re-entrant/thread safe.
     *
     * \note The idea behind this method is that you have guaranteed
     * access to the data of the object passed as argument. That is,
     * the occupied memory is not deallocated while your method is
     * running. You cannot safely access anything else unles it is
     * also a dynamically allocated object.
     */
    template<typename T>
    static Ref<Thread> run(Ref<T>, void (T::*)(), bool start = true);

    /**
     * Like Thread::run(Ref<T>, void (T::*)()) except that it
     * invokes a method taking one argument, and you must specify
     * that extra argument when calling this method.
     *
     * \note Be sure to not use an aliased types '&' for the method
     * argument, since copying is essential to thread safty.
     */
```



```
    template<typename T, typename U>
    static Ref<Thread> run(Ref<T>, void (T::*)(U), U, bool start = true);

    /**
     * Let a new thread invoke a static 1-argument-function of your
     * choise.
     *
     * \note This method is re-entrant/thread safe.
     *
     * \note Be sure to not use an aliased types '&' for the method
     * argument, since copying is essential to thread safty.
     *
     * \note You must not considder global data to be
     * acccessible. They may be destroyed.
     */
    template<typename T>
    static Ref<Thread> run(void (*)(T), T, bool start = true);

    /**
     * Let a new thread invoke a static void-function of your
     * choise.
     *
     * \note This method is re-entrant/thread safe.
     *
     * \note You must not considder global data to be
     * acccessible. They may be destroyed.
     */
    static Ref<Thread> run(void (*)(), bool start = true);

    /**
     * Let a new thread invoke the virtual main method of the
     * specified Thread object. To be of any use, you must derive
     * your own class from the Thread object. An extra reference
     * count is held on the thread object during the execution of
     * the specified method, and it is released afterwards.
     *
     * \note It is strongly recomended that you dont use this method
     * in the constructor specifying 'this' as argument. If you do,
     * you might get back fro the constructor an object which has
     * already been deleted.
     *
     * \sa Ref<T>
     *
     * \note This method is re-entrant/thread safe.
     *
     * \note NEVER do a Thread::start(this) in a constructor!
     *
     * \note The idea behind this methid is that you have guaranteed
     * access to the data of the object passed as argument. That is,
     * the occupied memory is not deallocated while your method is
     * running. You cannot safely access anything else unles it is
     * also a dynamically allocated object.
     */
    static void start(Ref<Thread>) throw(AlreadyStartedException);

    /**
     * Wait for this thread to stop.
     *
```

```
     * \note This method is re-entrant/thread safe. That is, any
     * number of threads may wait simultaniously.
     */
    void wait() throw(ThreadTerminatedException, NotStartedException);

    /**
     * Issue a termination request for this thread. This causes any
     * present or furure calls to
     *
     * Semaphore::down
     * Condition::wait,
     * Condition::timedWait,
     * Condition::select,
     * Thread::wait,
     * Thread::sleep,
     * Thread::acceptTermination,
     *
     * by this thread to imediately throw a
     * TerminatedException. This method will normally be called by
     * some other thread in the intent to bring the execution of
     * this thread to an end.
     *
     * \note This method is re-entrant/thread safe.
     */
    void terminate();

    /**
     * Return a reference to the Thread object associated with the
     * thread that calls this method. This also works for the main
     * thread and any other thread created in the context of the
     * pthead library.
     *
     * \note This method is re-entrant/thread safe.
     */
    static Ref<Thread> self();

    /**
     * Make the calling thread sleep for the specified amount of
     * time.
     *
     * \sa Time
     *
     * \note This method is re-entrant/thread safe.
     */
    static void sleep(const Time &period) throw(ThreadTerminatedException);

    /**
     * Make the calling thread sleep until the specified point in
     * time has been reached.
     *
     * \sa Time
     *
     * \note This method is re-entrant/thread safe.
     */
    static void sleepUntil(const Time &timeout) throw(
         ThreadTerminatedException);

    /**
```



```
     * This method should be called by a thread if it is executing
     * lengthy computations without any calls to Condition::wait,
     * Condition::timedWait or Thread::sleep. This is to ensure a
     * fair response time to a termination request by another
     * thread.  Note, the more frequent this method is called the
     * shorter the response to a tremination request will be.
     *
     * \note This method is re-entrant/thread safe.
     */
    static void acceptTermination() throw(ThreadTerminatedException);

    /**
     * Recover from termination. That is, you are allowed to catch
     * the TerminatedException in your thread if you do not want
     * your thread to terminate completely. But note that this may
     * be in conflict with the intent of some other thread calling
     * terminate on this thread.
     *
     * \note This method is re-entrant/thread safe.
     */
    static void selfResurrect();

    /**
     * When called by the main thread it suspends the main thread
     * until all other threads have terminated.
     *
     * The intention is that you should call this method just before
     * returning from the main function of your program. The
     * intended purpose of this method is enforce proper
     * deallocation of all resources by ensuring that all stacks are
     * unwinded completely before your program exits.
     *
     * It makes no sence to call this method for any other thread
     * than the main thread, since when the main thread terminates
     * it will imadiately annihilate all other threads, preventing
     * them from proper stack unwinding.
     *
     * Theads which are not created by one of the methods in this
     * Thread class and which have never called the Thread::self
     * method are unknown to this method, and will therefore not be
     * waited for.
     */
    static void mainExitWait();

  protected:
    /**
     * This is the method that will be run as the new thread. You
     * should override this method.
     */
    virtual void main() = 0;

    /**
     * Does not start the thread. You must use Thread::start to do
     * that - BUT NOT WITHIN THE COSTRUCTOR!
     */
    Thread();

    virtual ~Thread();
```

```
private:
  friend struct Condition;

  struct SelfThread;

  pthread_t pthread;
  bool started;
  bool terminated;
  Condition termination;

  volatile bool terminateRequest;
  Mutex waitConditionMutex;
  Condition *waitCondition;

  static void selfKeyDestroy(void *) throw();
  static void selfKeyAlloc() throw();
  static void registerSelf(Thread *);

  static void *entry(Thread *thread) throw();

  static void activate(Thread *);
  static void deactivate(Thread *);

  Thread(const Thread &); // Hide
  Thread &operator=(const Thread &); // Hide
};



/**
 * A semaphore.
 */
struct Semaphore
{
  Semaphore(int initialValue=1): value(initialValue), nonZero(mutex) {}

  /**
   * proberen/wait
   */
  void down() throw(ThreadTerminatedException);

  /**
   * verhogan/signal
   */
  void up();

private:
  int value;
  Mutex mutex;
  Condition nonZero;
};



template<typename T> struct MethodVoidRunner: Thread
{
```



```
  Ref<T> o;
  void (T::*m)();
  MethodVoidRunner(Ref<T> o, void (T::*m)()): o(o), m(m) {}
  void main()
  {
    Ref<T> p;
    p.swap(o);
    (p.get()->*m)();
  }
};

template<typename T>
Ref<Thread> Thread::run(Ref<T> o, void (T::*m)(), bool s)
{
  Ref<MethodVoidRunner<T> > r = new MethodVoidRunner<T>(o, m);
  if(s) Thread::start(r);
  return r;
}



template<typename T, typename U> struct MethodArgRunner: Thread
{
  Ref<T> o;
  void (T::*m)(U);
  U a;
  MethodArgRunner(Ref<T> o, void (T::*m)(U), U a): o(o), m(m), a(a) {}
  void main()
  {
    Ref<T> p;
    U b;
    std::swap(o, p);
    std::swap(a, b);
    (p.get()->*m)(b);
  }
};

template<typename T, typename U>
Ref<Thread> Thread::run(Ref<T> o, void (T::*m)(U), U a, bool s)
{
  Ref<MethodArgRunner<T, U> > r = new MethodArgRunner<T, U>(o, m, a);
  if(s) Thread::start(r);
  return r;
}



struct FuncVoidRunner: Thread
{
  void (*f)();
  FuncVoidRunner(void (*f)()): f(f) {}
  void main() { (*f)(); }
};

inline Ref<Thread> Thread::run(void (*f)(), bool s)
{
  Ref<FuncVoidRunner> r = new FuncVoidRunner(f);
  if(s) Thread::start(r);
```

```
      return r;
    }



    template<typename T> struct FuncArgRunner: Thread
    {
      void (*f)(T);
      T a;
      FuncArgRunner(void (*f)(T), T a): f(f), a(a) {}
      void main()
      {
        T b;
        std::swap(a, b);
        (*f)(b);
      }
    };

    template<typename T>
    Ref<Thread> Thread::run(void (*f)(T), T a, bool s)
    {
      Ref<FuncArgRunner<T> > r = new FuncArgRunner<T>(f, a);
      if(s) Thread::start(r);
      return r;
    }
  }
}

#endif // ARCHON_UTILITIES_THREAD_H
```

Listing: **archon/util/thread.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#include <errno.h>
#include <iostream>

#include <archon/util/thread.H>

namespace Archon
{
```



```
namespace Utilities
{
  using namespace std;

  namespace
  {
    /**
     * TSD key for the thread self reference.
     */
    pthread_key_t selfKey;

    /**
     * Once-only allocation of the thread self reference TSD key.
     */
    pthread_once_t selfKeyOnce = PTHREAD_ONCE_INIT;

    /**
     * Protects activeThreads and lastActiveThread.
     */
    pthread_mutex_t activeThreadsMutex = PTHREAD_MUTEX_INITIALIZER;

    /**
     * Signalled when activeThreads goes from 2 to 1.
     */
    pthread_cond_t lastActiveThread = PTHREAD_COND_INITIALIZER;

    /**
     * Number of running threads which have either been started
     * through one of the methods of Thread or have called
     * Thread::self at least once.
     */
    unsigned long activeThreads = 0;

    void lockMutex(pthread_mutex_t *m)
    {
      int e = pthread_mutex_lock(m);
      if(e == 0) return;
      if(e == EDEADLK)
        ARCHON_THROW1(Mutex::StateException,
                      "Attempt to lock a locked mutex");
      if(e == EINVAL)
        ARCHON_THROW1(Mutex::InternalException,
                      "Attempt to lock uninitialized mutex");
      ARCHON_THROW1(Mutex::InternalException,
                    "Attempt to lock mutex failed");
    }

    void unlockMutex(pthread_mutex_t *m)
    {
      int e = pthread_mutex_unlock(m);
      if(e == 0) return;
      if(e == EPERM)
        ARCHON_THROW1(Mutex::StateException,
                      "Attempt to unlock mutex that is locked by another
                           thread");
      if(e == EINVAL)
        ARCHON_THROW1(Mutex::InternalException,
                      "Attempt to unlock uninitialized mutex");
```

```
    ARCHON_THROW1(Mutex::InternalException,
                  "Attempt to unlock mutex failed");
  }
}


void Thread::activate(Thread *t)
{
  lockMutex(&activeThreadsMutex);
  ++activeThreads;
  unlockMutex(&activeThreadsMutex);
  t->refInc();
}

/**
 * It is important that the signalling on the lastThread
 * condition is the very last thing that happens for the thread,
 * since this ensures that all resources held by the thread is
 * deallocated before a call to Thread::mainExitWait returns. In
 * particular it is important that the usage count is
 * decremented before the signalling on lastThread, since if it
 * was not, and the self reference TSD key is the last reference
 * to the usage count then the thread object might never get
 * deallocated because of main thread exit.
 */
void Thread::deactivate(Thread *t)
{
  t->refDec();
  lockMutex(&activeThreadsMutex);
  bool l = --activeThreads == 1;
  unlockMutex(&activeThreadsMutex);
  if(l)
  {
    const int e = pthread_cond_broadcast(&lastActiveThread);
    if(e != 0)
      ARCHON_THROW1(Condition::InternalException,
                    "Attempt to broadcast on condition failed");
  }
}

/**
 * Handle destruction of self reference TSD key.
 *
 * This is also the method that is responsible for decrementing
 * the usage count of the associated thread object.
 */
void Thread::selfKeyDestroy(void *v) throw()
{
  Thread *t = reinterpret_cast<Thread *>(v);
  {
    Mutex::Lock l(*t);
    t->terminated = true;
  }
  t->termination.notifyAll();
  deactivate(t);
}

/**
```



```
 * Allocate the thread self reference TSD key.
 */
void Thread::selfKeyAlloc() throw()
{
  int error = pthread_key_create(&selfKey, &Thread::selfKeyDestroy);
  if(error)
  {
    if(error == EAGAIN)
      ARCHON_THROW1(Thread::ResourceException,
                    "Too many TSD keys");
    ARCHON_THROW1(Thread::InternalException,
                  "Could not create new TSD key");
  }
}

/**
 * Must be called with one extra reference count.
 */
void Thread::registerSelf(Thread *t)
{
  int error = pthread_once(&selfKeyOnce, &selfKeyAlloc);
  if(error)
  {
    deactivate(t);
    ARCHON_THROW1(InternalException, "pthread_once failed");
  }
  error = pthread_setspecific(selfKey, reinterpret_cast<void *>(t));
  if(error)
  {
    deactivate(t);

    if(error == EINVAL)
      ARCHON_THROW1(InternalException,
                    "Invalid TSD key in pthread_setspecific");
    ARCHON_THROW1(InternalException,
                  "pthread_setspecific failed");
  }
}

Thread::Thread():
  started(false), terminated(false), termination(*this),
  terminateRequest(false), waitCondition(0)
{
}

Thread::~Thread()
{{ // The extra scope is needed to work around gcc3.2 bug #8287
}}

/**
 * Must be called with one extra reference count.
 */
void *Thread::entry(Thread *t) throw()
{
  int error = pthread_setcancelstate(PTHREAD_CANCEL_DISABLE, 0);
  if(error) ARCHON_THROW1(ResourceException,
                          "pthread_setcancelstate failed");
  registerSelf(t);
```

```
  try
  {
    t->main();
  }
  catch(ThreadTerminatedException &) {}
  return 0;
}

void Thread::start(Ref<Thread> t) throw(AlreadyStartedException)
{
  if(!t) ARCHON_THROW1(ArgumentException, "Got null thread");
  {
    Mutex::Lock l(*t.get());
    if(t->started) ARCHON_THROW(AlreadyStartedException);
    t->started = true;
  }

  pthread_attr_t attr;
  int error = pthread_attr_init(&attr);
  if(error) ARCHON_THROW1(ResourceException, "pthread_attr_init failed");
  error = pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
  if(error) ARCHON_THROW1(ResourceException,
                          "pthread_attr_setdetachstate failed");

  activate(t.get());
  error = pthread_create
    (&t->pthread, 0, reinterpret_cast<void *(*)(void *)>(t->entry), t.get
         ());
  pthread_attr_destroy(&attr);

  if(!error) return;
  deactivate(t.get());
  if(error == EAGAIN)
    ARCHON_THROW1(ResourceException,
                  "Not enough system resources to create a new thread");
  ARCHON_THROW1(InternalException,
                "Could not create a new thread");
}


struct Thread::SelfThread: Thread
{
  SelfThread()
  {
    pthread = pthread_self();
    started = true;
  }

  void main() {} // Dummy
};

Ref<Thread> Thread::self()
{
  int error = pthread_once(&selfKeyOnce, &selfKeyAlloc);
  if(error) ARCHON_THROW1(InternalException, "pthread_once failed");
  Thread *t = reinterpret_cast<Thread *>(pthread_getspecific(selfKey));
  if(t) return t;
  t = new SelfThread;
```



```
  activate(t);
  registerSelf(t);
  return t;
}

void Thread::terminate()
{
  terminateRequest = true;
  Mutex::Lock l1(waitConditionMutex);
  if(waitCondition)
  {
    Mutex::Lock l2(*waitCondition->mutex);
    waitCondition->notifyAll();
  }
  l1.release();
}

void Thread::selfResurrect()
{
  self()->terminateRequest = false;
}

void Thread::wait() throw(ThreadTerminatedException, NotStartedException)
{
  Mutex::Lock l(*this);
  if(!started) ARCHON_THROW(NotStartedException);
  while(!terminated) termination.wait();
}

void Thread::sleep(const Time &period) throw(ThreadTerminatedException)
{
  Time timeout(Time::now());
  timeout += period;
  Mutex mutex;
  Condition cond(mutex);
  Mutex::Lock lock(mutex);
  while(!cond.timedWait(timeout));
}

void Thread::sleepUntil(const Time &timeout) throw(
     ThreadTerminatedException)
{
  Mutex mutex;
  Condition cond(mutex);
  Mutex::Lock lock(mutex);
  while(!cond.timedWait(timeout));
}

void Thread::acceptTermination() throw(ThreadTerminatedException)
{
  if(self()->terminateRequest) ARCHON_THROW(ThreadTerminatedException);
}

void Thread::mainExitWait()
{
  Thread::self(); // Must make sure that the calling thread counts
                  // as active.
  lockMutex(&activeThreadsMutex);
```

```
      while(activeThreads > 1)
        if(pthread_cond_wait(&lastActiveThread, &activeThreadsMutex))
          ARCHON_THROW1(Condition::InternalException, "pthread_cond_wait
               failed");
      unlockMutex(&activeThreadsMutex);
    }


    void Semaphore::down() throw(ThreadTerminatedException)
    {
      Mutex::Lock l(mutex);
      while(value == 0) nonZero.wait();
      --value;
    }

    void Semaphore::up()
    {
      Mutex::Lock l(mutex);
      ++value;
      l.release();
      nonZero.notifyAll();
    }
  }
}
```

Listing: **archon/x3d/proxy/application.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_PROXY_APPLICATION_H
#define ARCHON_X3D_PROXY_APPLICATION_H

#include <archon/x3d/proxy/context.H>

namespace Archon
{
  namespace X3D
  {
    namespace Proxy
    {
      struct Session;
```



```
      struct Application: ExecutionContext
      {
        static Ref<Application> create(Ref<Session> s, Uri baseUri)
        {
          return s->createApplication(baseUri);
        }

        static Ref<Application> load(Ref<Session> s, Uri uri)
        {
          return s->loadApplication(uri);
        }

        void launch();

      private:
        friend struct Session;

        Application(Session *s, unsigned long id): ExecutionContext(s, id) {}
      };
    }
  }
}

#endif // ARCHON_X3D_PROXY_APPLICATION_H
```

Listing: **archon/x3d/proxy/application.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#include <archon/x3d/proxy/application.H>

namespace Archon
{
  namespace X3D
  {
    namespace Proxy
    {
      void Application::launch()
      {
        session->session->launchApplicationScene(id);
```

```
      }
    }
  }
}
```

Listing: **archon/x3d/proxy/autogen.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#include <fstream>

#include <archon/x3d/server/x3d.H>

using namespace Archon::X3D;

map<const FieldType *, string> retTypes;
map<const FieldType *, string> argTypes;

int indentationStep;

inline string indent(int level)
{
  int n = level * indentationStep;
  int m = 0;
  while(n>=8) { n -= 8; ++m; }
  return string(m, '\t') + string(n, ' ');
}

int main(int argc, const char *argv[])
{
  indentationStep = argc > 1 ? atoi(argv[1]) : 2;
  string config_H = argc > 2 ? argv[2] : "config.autogen.H";
  string type_C   = argc > 3 ? argv[3] : "type.autogen.C";
  string node_H   = argc > 4 ? argv[4] : "node.autogen.H";

  Archon::X3D::initialize();

  retTypes[SFBool::type] = "bool ";
  retTypes[MFBool::type] = "std::vector<bool> ";

  retTypes[SFColor::type] = "Vector3 ";
```



```
  retTypes[MFColor::type] = "std::vector<Vector3> ";

  retTypes[SFColorRGBA::type] = "Vector4 ";
  retTypes[MFColorRGBA::type] = "std::vector<Vector4> ";

  retTypes[SFDouble::type] = "double ";
  retTypes[MFDouble::type] = "std::vector<double> ";

  retTypes[SFFloat::type] = "double ";
  retTypes[MFFloat::type] = "std::vector<double> ";

  retTypes[SFImage::type] = "Image ";
  retTypes[MFImage::type] = "std::vector<Image> ";

  retTypes[SFInt32::type] = "int ";
  retTypes[MFInt32::type] = "std::vector<int> ";

  retTypes[SFRotation::type] = "AxisRot3 ";
  retTypes[MFRotation::type] = "std::vector<AxisRot3> ";

  retTypes[SFString::type] = "std::string ";
  retTypes[MFString::type] = "std::vector<std::string> ";

  retTypes[SFTime::type] = "Time ";
  retTypes[MFTime::type] = "std::vector<Time> ";

  retTypes[SFVec2d::type] = "Vector2 ";
  retTypes[MFVec2d::type] = "std::vector<Vector2> ";

  retTypes[SFVec2f::type] = "Vector2 ";
  retTypes[MFVec2f::type] = "std::vector<Vector2> ";

  retTypes[SFVec3d::type] = "Vector3 ";
  retTypes[MFVec3d::type] = "std::vector<Vector3> ";

  retTypes[SFVec3f::type] = "Vector3 ";
  retTypes[MFVec3f::type] = "std::vector<Vector3> ";


  argTypes[SFBool::type] = "bool ";
  argTypes[MFBool::type] = "const std::vector<bool> &";

  argTypes[SFColor::type] = "Vector3 ";
  argTypes[MFColor::type] = "const std::vector<Vector3> &";

  argTypes[SFColorRGBA::type] = "Vector4 ";
  argTypes[MFColorRGBA::type] = "const std::vector<Vector4> &";

  argTypes[SFDouble::type] = "double ";
  argTypes[MFDouble::type] = "const std::vector<double> &";

  argTypes[SFFloat::type] = "double ";
  argTypes[MFFloat::type] = "const std::vector<double> &";

  argTypes[SFImage::type] = "const Image &";
  argTypes[MFImage::type] = "const std::vector<Image> &";

  argTypes[SFInt32::type] = "int ";
```

```
argTypes[MFInt32::type] = "const std::vector<int> &";

argTypes[SFRotation::type] = "AxisRot3 ";
argTypes[MFRotation::type] = "const std::vector<AxisRot3> &";

argTypes[SFString::type] = "std::string ";
argTypes[MFString::type] = "const std::vector<std::string> &";

argTypes[SFTime::type] = "Time ";
argTypes[MFTime::type] = "const std::vector<Time> &";

argTypes[SFVec2d::type] = "Vector2 ";
argTypes[MFVec2d::type] = "const std::vector<Vector2> &";

argTypes[SFVec2f::type] = "Vector2 ";
argTypes[MFVec2f::type] = "const std::vector<Vector2> &";

argTypes[SFVec3d::type] = "Vector3 ";
argTypes[MFVec3d::type] = "const std::vector<Vector3> &";

argTypes[SFVec3f::type] = "Vector3 ";
argTypes[MFVec3f::type] = "const std::vector<Vector3> &";


// node.H

{
  int l = 0;

  ofstream out(node_H.c_str());

  out
    << indent(l) << "/*\n"
    << indent(l) << " * This file is part of the \"Archon\" framework.\n"
    << indent(l) << " * (http://files3d.sourceforge.net)\n"
    << indent(l) << " *\n"
    << indent(l) << " * Copyright © 2002 by Kristian Spangsege and Brian
         Kristiansen.\n"
    << indent(l) << " *\n"
    << indent(l) << " * Permission to use, copy, modify, and distribute
         this software and\n"
    << indent(l) << " * its documentation under the terms of the GNU
         General Public License is\n"
    << indent(l) << " * hereby granted. No representations are made about
         the suitability of\n"
    << indent(l) << " * this software for any purpose. It is provided \"as
         is\" without express\n"
    << indent(l) << " * or implied warranty. See the GNU General Public
         License\n"
    << indent(l) << " * (http://www.gnu.org/copyleft/gpl.html) for more
         details.\n"
    << indent(l) << " *\n"
    << indent(l) << " * The characters in this file are ISO8859-1 encoded.\
         n"
    << indent(l) << " *\n"
    << indent(l) << " * The documentation in this file is in \"Doxygen\"
         style\n"
    << indent(l) << " * (http://www.doxygen.org).\n"
```



```
  << indent(l) << " */\n"
  << "\n"
  << indent(l) << "/*\n"
  << indent(l) << " * Note: This file is auto-generated, so don't edit it
       !\n"
  << indent(l) << " */\n"
  << "\n"
  << "#ifndef ARCHON_X3D_PROXY_NODE_AUTOGEN_H\n"
  << "#define ARCHON_X3D_PROXY_NODE_AUTOGEN_H\n"
  << "\n"
  << "#include <archon/x3d/proxy/field_type.H>\n"
  << "#include <archon/x3d/proxy/field.H>\n"
  << "\n"
  << indent(l) << "namespace Archon\n"
  << indent(l) << "{\n"
  << indent(l+1) << "namespace X3D\n"
  << indent(l+1) << "{\n"
  << indent(l+1+1) << "namespace Proxy\n"
  << indent(l+1+1) << "{\n";

l += 3;

for(unsigned long i=0; i<NodeType::getTotalNumber(); ++i)
{
  const NodeType *t = NodeType::get(i);

  if(i) out
    << "\n"
    << "\n"
    << "\n";

  out
    << indent(l) << "/**\n"
    << indent(l) << " * " << (t->isAbstract() ? "Abstract" : "Concrete"
         ) << " X3D node" << (t->isVirtualBase() ? " (virtual base)" : ""
         ) << "\n"
    << indent(l) << " */\n"
    << indent(l) << "struct " << t->getName();

  if(t==Node::type) out
    << ": virtual NodeBase";
  else
  {
    NodeType::iterator j = t->derivedFromBegin();
    if(j != t->derivedFromEnd())
    {
      out
        << ": ";
      while(j != t->derivedFromEnd())
      {
        const NodeType *d = *j;
        if(j != t->derivedFromBegin()) out
          << ", ";
        if(d->isVirtualBase()) out
          << "virtual ";
        out
          << d->getName();
        ++j;
```

```
    }
  }
}

out
  << "\n"
  << indent(l) << "{\n";

if(!t->isAbstract())
{
  out
    << indent(l+1) << "static Ref<" << t->getName() << "> create(Ref<
         ExecutionContext> c)\n"
    << indent(l+1) << "{\n"
    << indent(l+1+1) << "return NodeBase::create<" << t->getName() <<
         ">(c.get(), " + Archon::Utilities::Text::toString(i) + ");\n"
    << indent(l+1) << "}\n"
    << "\n";
}

for(unsigned long j=0; j<t->getNumberOfImmediateFields(); ++j)
{
  const FieldBase *f = t->getImmediateField(j);

  string name = f->getName();

  string access =
    f->getIsEventSource() ?
    f->getIsEventTarget() ? "InputOutput" : "OutputOnly" :
    f->getIsEventTarget() ? "InputOnly" : "InitializeOnly";

  if(f->getType()->getIsMultiple())
    access += "Seq";

  string type =
    f->getType()->getName();

  if(const NodeFieldBase *nf = dynamic_cast<const NodeFieldBase *>(f))
    type += "<" + nf->getNodeType()->getName() + "> ";

  out
    << indent(l+1) << access << "<" << type << "> "
    << name << ";\n";
}

if(t->getNumberOfImmediateFields()) out
  << "\n";

for(unsigned long j=0; j<t->getNumberOfImmediateFields(); ++j)
{
  const FieldBase *f = t->getImmediateField(j);

  string name = f->getName();

  // Skip special fields
  if(f->getFlags()) continue;

  out
```



```
    << indent(l+1) << "/*\n"
    << indent(l+1) << " * " << name << "\n"
    << indent(l+1) << " */\n";

  // Determine field type
  string cap = Archon::Utilities::Text::initCap(name);
  string set = "set" + cap;
  if(const NodeSequenceFieldBase *f2 =
     dynamic_cast<const NodeSequenceFieldBase *>(f))
  {
    string child = f2->getNodeType()->getName();
    if(f->getIsEventSource() || !f->getIsEventTarget()) out
      << indent(l+1) << "std::vector<Ref<" << child << "> > get" << cap
           << "()\n"
      << indent(l+1) << "{\n"
      << indent(l+1+1) << "return " + name + ";\n"
      << indent(l+1) << "}\n";
    if(!f->getIsEventSource() || f->getIsEventTarget()) out
      << indent(l+1) << "void " << set << "(const std::vector<Ref<" +
           child + "> > &v)\n"
      << indent(l+1) << "{\n"
      << indent(l+1+1) << name + " = v;\n"
      << indent(l+1) << "}\n";
  }
  else if(const NodeFieldBase *f2 =
          dynamic_cast<const NodeFieldBase *>(f))
  {
    string child = f2->getNodeType()->getName();
    if(f->getIsEventSource() || !f->getIsEventTarget()) out
      << indent(l+1) << "Ref<" + child + "> get" << cap << "()\n"
      << indent(l+1) << "{\n"
      << indent(l+1+1) << "return " + name + ";\n"
      << indent(l+1) << "}\n";
    if(!f->getIsEventSource() || f->getIsEventTarget()) out
      << indent(l+1) << "void " << set << "(Ref<" + child + "> v)\n"
      << indent(l+1) << "{\n"
      << indent(l+1+1) << name + " = v;\n"
      << indent(l+1) << "}\n";
  }
  else
  {
    string ret = retTypes[f->getType()];
    string arg = argTypes[f->getType()];
    string type = f->getType()->getName();
    if(f->getIsEventSource() || !f->getIsEventTarget()) out
      << indent(l+1) << ret << "get" << cap << "()\n"
      << indent(l+1) << "{\n"
      << indent(l+1+1) << "return " + name + ";\n"
      << indent(l+1) << "}\n";
    if(!f->getIsEventSource() || f->getIsEventTarget()) out
      << indent(l+1) << "void " << set << "(" << arg << "v)\n"
      << indent(l+1) << "{\n"
      << indent(l+1+1) << name + " = v;\n"
      << indent(l+1) << "}\n";
  }
  out
    << "\n";
}
```

```
    string fieldInit;
    for(unsigned long j=0; j<t->getNumberOfImmediateFields(); ++j)
    {
      const FieldBase *f = t->getImmediateField(j);

      string name = f->getName();

      fieldInit += string(j||!t->isAbstract()?",":":") + "\n" + indent(l
           +1+1) + name + "(this, " + Archon::Utilities::Text::toString(f->
           getId()) + ")";
    }

    if(t->isAbstract())
    {
      out
        << indent(l) << "protected:\n"
        << indent(l+1) << t->getName() << "()" << fieldInit << " {}\n";
    }
    else
    {
      out
        << indent(l) << "private:\n"
        << indent(l+1) << t->getName() << "(Session *s, unsigned long i):
            NodeBase(s, i)" << fieldInit << " {}\n"
        << "\n"
        << indent(l) << "public:\n"
        << indent(l+1) << "static NodeBase *instantiate(Session *s,
            unsigned long i)\n"
        << indent(l+1) << "{\n"
        << indent(l+1+1) << "return new " << t->getName() << "(s, i);\n"
        << indent(l+1) << "}\n";
    }

    out
      << indent(l) << "};\n";

    cout << t->getName() << "\n";
  }

  l -= 3;

  out
    << indent(l+1+1) << "}\n"
    << indent(l+1) << "}\n"
    << indent(l) << "}\n"
    << "\n"
    << "#endif // ARCHON_X3D_PROXY_NODE_AUTOGEN_H\n";
}


// config.H

{
  int l = 0;

  ofstream out(config_H.c_str());
```



```
  out
    << indent(l) << "/*\n"
    << indent(l) << " * This file is part of the \"Archon\" framework.\n"
    << indent(l) << " * (http://files3d.sourceforge.net)\n"
    << indent(l) << " *\n"
    << indent(l) << " * Copyright © 2002 by Kristian Spangsege and Brian
         Kristiansen.\n"
    << indent(l) << " *\n"
    << indent(l) << " * Permission to use, copy, modify, and distribute
         this software and\n"
    << indent(l) << " * its documentation under the terms of the GNU
         General Public License is\n"
    << indent(l) << " * hereby granted. No representations are made about
         the suitability of\n"
    << indent(l) << " * this software for any purpose. It is provided \"as
         is\" without express\n"
    << indent(l) << " * or implied warranty. See the GNU General Public
         License\n"
    << indent(l) << " * (http://www.gnu.org/copyleft/gpl.html) for more
         details.\n"
    << indent(l) << " *\n"
    << indent(l) << " * The characters in this file are ISO8859-1 encoded.\
         n"
    << indent(l) << " *\n"
    << indent(l) << " * The documentation in this file is in \"Doxygen\"
         style\n"
    << indent(l) << " * (http://www.doxygen.org).\n"
    << indent(l) << " */\n"
    << "\n"
    << indent(l) << "/*\n"
    << indent(l) << " * Note: This file is auto-generated, so don't edit it
         !\n"
    << indent(l) << " */\n"
    << "\n"
    << "#ifndef ARCHON_X3D_PROXY_CONFIG_AUTOGEN_H\n"
    << "#define ARCHON_X3D_PROXY_CONFIG_AUTOGEN_H\n"
    << "\n"
    << indent(l) << "namespace Archon\n"
    << indent(l) << "{\n"
    << indent(l+1) << "namespace X3D\n"
    << indent(l+1) << "{\n"
    << indent(l+1+1) << "namespace Proxy\n"
    << indent(l+1+1) << "{\n"
    << indent(l+1+1+1) << "const unsigned long numberOfNodeTypes = " <<
         NodeType::getTotalNumber() << ";\n"
    << indent(l+1+1) << "}\n"
    << indent(l+1) << "}\n"
    << indent(l) << "}\n"
    << "\n"
    << "#endif // ARCHON_X3D_PROXY_CONFIG_AUTOGEN_H\n";
}



// type.C

{
  int l = 0;
```

```
ofstream out(type_C.c_str());

out
  << indent(l) << "/*\n"
  << indent(l) << " * This file is part of the \"Archon\" framework.\n"
  << indent(l) << " * (http://files3d.sourceforge.net)\n"
  << indent(l) << " *\n"
  << indent(l) << " * Copyright © 2002 by Kristian Spangsege and Brian
       Kristiansen.\n"
  << indent(l) << " *\n"
  << indent(l) << " * Permission to use, copy, modify, and distribute
       this software and\n"
  << indent(l) << " * its documentation under the terms of the GNU
       General Public License is\n"
  << indent(l) << " * hereby granted. No representations are made about
       the suitability of\n"
  << indent(l) << " * this software for any purpose. It is provided \"as
       is\" without express\n"
  << indent(l) << " * or implied warranty. See the GNU General Public
       License\n"
  << indent(l) << " * (http://www.gnu.org/copyleft/gpl.html) for more
       details.\n"
  << indent(l) << " *\n"
  << indent(l) << " * The characters in this file are ISO8859-1 encoded.\
       n"
  << indent(l) << " *\n"
  << indent(l) << " * The documentation in this file is in \"Doxygen\"
       style\n"
  << indent(l) << " * (http://www.doxygen.org).\n"
  << indent(l) << " */\n"
  << "\n"
  << indent(l) << "/*\n"
  << indent(l) << " * Note: This file is auto-generated, so don't edit it
       !\n"
  << indent(l) << " */\n"
  << "\n"
  << "#include <archon/x3d/proxy/type.H>\n"
  << "#include <archon/x3d/proxy/" << node_H << ">\n"
  << "\n"
  << indent(l) << "namespace Archon\n"
  << indent(l) << "{\n"
  << indent(l+1) << "namespace X3D\n"
  << indent(l+1) << "{\n"
  << indent(l+1+1) << "namespace Proxy\n"
  << indent(l+1+1) << "{\n";

l += 3;

for(unsigned long i=0; i<NodeType::getTotalNumber(); ++i)
{
  const NodeType *t = NodeType::get(i);

  if(!t->getNumberOfImmediateFields()) continue;

  out
    << indent(l) << "const FieldDef fieldsOf" + t->getName() + "[] =\n"
    << indent(l) << "{\n";
```



```
    l += 1;

    for(unsigned long j=0; j<t->getNumberOfImmediateFields(); ++j)
    {
      const FieldBase *f = t->getImmediateField(j);

      out
        << indent(l) << "{ \"" << f->getName() << "\" },\n";
    }

    l -= 1;

    out
      << indent(l) << "};\n"
      << "\n";
  }


  out
    << indent(l) << "const NodeDef nodeDefs[] =\n"
    << indent(l) << "{\n";

  l += 1;

  for(unsigned long i=0; i<NodeType::getTotalNumber(); ++i)
  {
    const NodeType *t = NodeType::get(i);

    out
      << indent(l) << "{ " << "\"" << t->getName() << "\"";

    if(t->isUserInstantiable()) out
      << ", &" << t->getName() << "::instantiate";
    else out
      << ", 0";

    out
      << ", " << t->getNumberOfImmediateFields();

    if(t->getNumberOfImmediateFields()) out
      << ", fieldsOf" + t->getName();
    else out
      << ", 0";

    out
      << " },\n";
  }

  l -= 4;

  out
    << indent(l+1+1+1) << "};\n"
    << indent(l+1+1) << "}\n"
    << indent(l+1) << "}\n"
    << indent(l) << "}\n";
}
```

```
  return 0;
}
```

Listing: archon/x3d/proxy/context.H

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_PROXY_CONTEXT_H
#define ARCHON_X3D_PROXY_CONTEXT_H

#include <archon/util/uri.H>

#include <archon/x3d/proxy/session.H>

namespace Archon
{
  namespace X3D
  {
    namespace Proxy
    {
      struct Session;
      struct NodeBase;
      struct Node;
      struct Group;
      struct Field;

      template<typename F> struct EventSource;
      template<typename F> struct EventTarget;

      struct ExecutionContext: virtual RefObjectBase
      {
        Ref<Session> getSession() const { return session; }

        Uri getBaseUri() const;
        Ref<Group> getRootGroup() const;
        void setRootGroup(Ref<Group>);
        Ref<Node> getNamedNode(string name) const;
        //Ref<GroupingNode> createGroupFromStream(Ref<Stream::Reader> r);

        ~ExecutionContext() {{ session->destroyContext(this); }}
```



```
        template<typename F, typename G>
        void createRoute(EventSource<F> &s, EventTarget<G> &t)
        {
          route(s, t, false);
        }

        template<typename F, typename G>
        void removeRoute(EventSource<F> &s, EventTarget<G> &t)
        {
          route(s, t, true);
        }

      protected:
        friend struct NodeBase;
        friend struct Session;

        ExecutionContext(Session *s, unsigned long id):
          session(s), id(id), transCount(0) {}

        Ref<Session> session;

        unsigned long id;
        unsigned long transCount;

      private:
        template<typename T> static void checkDerivation(T *) {}

        template<typename F, typename G>
        void route(EventSource<F> &s, EventTarget<G> &t, bool destroy)
        {
          if(false) { F *p; checkDerivation<G>(p); }
          route(&s, &t, destroy);
        }

        template<typename C, typename D>
        void route(EventSource<SFNode<C> > &s, EventTarget<SFNode<D> > &t,
             bool destroy)
        {
          if(false) { C *p; checkDerivation<D>(p); }
          route(&s, &t, destroy);
        }

        template<typename C, typename D>
        void route(EventSource<MFNode<C> > &s, EventTarget<MFNode<D> > &t,
             bool destroy)
        {
          if(false) { C *p; checkDerivation<D>(p); }
          route(&s, &t, destroy);
        }

        void route(Field *s, Field *t, bool destroy);
      };
    }
  }
}

#endif // ARCHON_X3D_PROXY_CONTEXT_H
```

Listing: **archon/x3d/proxy/context.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#include <archon/x3d/proxy/context.H>

#include <archon/x3d/proxy/node.autogen.H>

namespace Archon
{
  namespace X3D
  {
    namespace Proxy
    {
      Uri ExecutionContext::getBaseUri() const
      {
        CORBA::String_var u = session->session->getBaseUri(id);
        try
        {
          return Uri(string(u));
        }
        catch(Uri::SyntaxException &)
        {
          ARCHON_THROW1(Session::ProtocolException, "Got invalid URI");
        }
      }

      Ref<Group> ExecutionContext::getRootGroup() const
      {
        Ref<NodeBase> n =
          session->id2node(session->session->getRootGroup(id));
        if(!n) return 0;
        Group *g = dynamic_cast<Group *>(n.get());
        if(!g) ARCHON_THROW1(Session::ProtocolException,
                             "Context root group was not a Group");
        return g;
      }

      void ExecutionContext::setRootGroup(Ref<Group> g)
      {
        session->session->setRootGroup(id, g->id);
      }
```



```
      Ref<Node> ExecutionContext::getNamedNode(string name) const
      {
        Ref<NodeBase> n =
          session->id2node(session->session->getNode(id, name.c_str()));
        if(!n) return 0;
        Node *m = dynamic_cast<Node *>(n.get());
        if(!m) ARCHON_THROW1(Session::ProtocolException,
                             "Named node was not a Node");
        return m;
      }

      void ExecutionContext::route(Field *s, Field *t, bool destroy)
      {
        if(destroy)
          session->session->delRoute(id,
                                     s->self->id, s->id,
                                     t->self->id, t->id);
        else
          session->session->addRoute(id,
                                     s->self->id, s->id,
                                     t->self->id, t->id);
      }
    }
  }
}
```

Listing: **archon/x3d/proxy/disposer.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_PROXY_DISPOSER_H
#define ARCHON_X3D_PROXY_DISPOSER_H

#include <archon/util/thread.H>

#include <archon/x3d/proxy/sai.H>

namespace Archon
{
  namespace X3D
```

```
  {
    namespace Proxy
    {
      using namespace std;
      using namespace Utilities;

      /**
       * This class implements an asynchroniously executing thread
       * that is responsible for pushing information to the server
       * about the disposal of ExecutionContexts, Nodes and Routes. If
       * this was not done asynchroniously, then every destruction of
       * a reference by the main application thread (or threads) could
       * potentially involve a call to the server. This is not
       * fortunate since the server call is not a fast operation, and
       * since it happens at unpredictable times from the point of
       * view of the main application thread.
       */
      struct Disposer: Thread
      {
        Disposer(x3d::sai::Session_ptr);
        virtual ~Disposer();

        void addContext(unsigned long id, unsigned long transCount);
        void addNode(unsigned long id, unsigned long transCount);

      private:
        x3d::sai::Session_var session;

        void main();

        Mutex disposerMutex;
        Condition more;

        struct Entry
        {
          unsigned long id;
          unsigned long transCount;
          Entry(unsigned long id, unsigned long transCount):
            id(id), transCount(transCount) {}
        };

        vector<Entry> contexts;
        vector<Entry> nodes;
      };
    }
  }
}

#endif // ARCHON_X3D_PROXY_DISPOSER_H
```

Listing: **archon/x3d/proxy/disposer.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
```



```
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#include <iostream>

#include <archon/x3d/proxy/disposer.H>

namespace Archon
{
  namespace X3D
  {
    namespace Proxy
    {
      Disposer::Disposer(x3d::sai::Session_ptr ref):
        more(disposerMutex)
      {
        session = x3d::sai::Session::_duplicate(ref);
      }

      Disposer::~Disposer()
      {
      }

      void Disposer::addContext(unsigned long id, unsigned long transCount)
      {
        {
          Mutex::Lock l(disposerMutex);
          contexts.push_back(Entry(id, transCount));
        }
        more.notifyAll();
      }


      void Disposer::addNode(unsigned long id, unsigned long transCount)
      {
        {
          Mutex::Lock l(disposerMutex);
          nodes.push_back(Entry(id, transCount));
        }
        more.notifyAll();
      }

      void Disposer::main()
      {
        for(;;)
        {
          vector<Entry> c, n;
          {
```

```
            Mutex::Lock l(disposerMutex);
            while(!contexts.size() && !nodes.size()) more.wait();
            swap(contexts, c);
            swap(nodes, n);
          }

          x3d::sai::DisposeSeq cs(c.size());
          cs.length(c.size());
          for(unsigned i=0; i<c.size(); ++i)
          {
            Entry &e = c[i];
            x3d::sai::DisposeEntry &f = cs[i];
            f.id = e.id;
            f.transCount = e.transCount;
          }

          x3d::sai::DisposeSeq ns(n.size());
          ns.length(n.size());
          for(unsigned i=0; i<n.size(); ++i)
          {
            Entry &e = n[i];
            x3d::sai::DisposeEntry &f = ns[i];
            f.id = e.id;
            f.transCount = e.transCount;
          }

          session->disposeBatch(cs, ns);
        }
      }
    }
  }
}
```

Listing: **archon/x3d/proxy/event.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_PROXY_EVENT_H
#define ARCHON_X3D_PROXY_EVENT_H

#include <archon/util/uri.H>
```



```
#include <archon/x3d/proxy/session.H>

namespace Archon
{
  namespace X3D
  {
    namespace Proxy
    {
      struct EventHandlerBase: RefObjectBase
      {
      protected:
        friend struct Session;
        virtual void handle(Session *, const x3d::sai::Value &) = 0;
      };

      template<typename F>
      struct EventHandler: EventHandlerBase
      {
        virtual void handle(typename F::ArgType) = 0;

      private:
        void handle(Session *s, const x3d::sai::Value &w)
        {
          typename F::Type v;
          s->im(F::index, w, reinterpret_cast<void *>(&v));
          handle(v);
        }
      };

      template<typename C>
      struct EventHandler<SFNode<C> >: EventHandlerBase
      {
        typedef SFNode<C> F;

        virtual void handle(typename F::ArgType) = 0;

      private:
        void handle(Session *s, x3d::sai::Value &w)
        {
          Ref<NodeBase> v;
          s->im(F::index, w, reinterpret_cast<void *>(&v));
          C *c = dynamic_cast<C *>(v.get());
          if(!c) ARCHON_THROW1(Session::ProtocolException,
                               "Incompatible node type");
          handle(v);
        }
      };

      template<typename C>
      struct EventHandler<MFNode<C> >: EventHandlerBase
      {
        typedef MFNode<C> F;

        virtual void handle(typename F::ArgType) = 0;

      private:
        void handle(Session *s, x3d::sai::Value &w)
```

```
        {
          vector<Ref<NodeBase> > v;
          s->im(F::index, w, reinterpret_cast<void *>(&v));
          vector<Ref<C> > u(v.size());
          for(unsigned i=0; i<v.size(); ++i)
          {
            C *c = dynamic_cast<C *>(v[i].get());
            if(!c) ARCHON_THROW1(Session::ProtocolException,
                                 "Incompatible node type");
            u[i].reset(c);
          }
          handle(v);
        }
      };

      template<typename F>
      struct FuncCaller: EventHandler<F>
      {
        void (*f)(typename F::ArgType);
        FuncCaller(void (*f)(typename F::ArgType)): f(f) {}

      private:
        void handle(typename F::ArgType a)
        {
          (*f)(a);
        }
      };

      template<typename F, typename T>
      struct CookieFuncCaller: EventHandler<F>
      {
        void (*f)(typename F::ArgType, T);
        T cookie;
        CookieFuncCaller(void (*f)(typename F::ArgType, T), T cookie):
          f(f), cookie(cookie) {}

      private:
        void handle(typename F::ArgType a)
        {
          (*f)(a, cookie);
        }
      };
    }
  }
}

#endif // ARCHON_X3D_PROXY_EVENT_H
```

Listing: **archon/x3d/proxy/exception.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
```



```
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_PROXY_EXCEPTION_H
#define ARCHON_X3D_PROXY_EXCEPTION_H

#include <string>

#include <archon/util/exception.H>

namespace Archon
{
  namespace X3D
  {
    namespace Proxy
    {
      using namespace std;

      struct Exception: virtual Utilities::Exception
      {
        /**
         * \sa Utilities::Exception::catchInfo
         */
        static bool catchInfo(Formatter format, string &result);

        /**
         * \sa Utilities::Exception::terminal
         */
        static void terminal();

        static string catchShortExplanation();
      };


      /**
       * A problem with one or more of the method arguments
       */
      struct ArgumentException: virtual Exception, Utilities::
          ArgumentException
      {
        ArgumentException() {}
        ArgumentException(string l): Utilities::Exception(l) {}
        ArgumentException(string l, string m): Utilities::Exception(l, m) {}
      };

      /**
       * A problem with getting access to external resources
       */
      struct ResourceException: virtual Exception, Utilities::
          ResourceException
```

```
      {
        ResourceException() {}
        ResourceException(string l): Utilities::Exception(l) {}
        ResourceException(string l, string m): Utilities::Exception(l, m) {}
      };

      /**
       * The request operation is not compatible with the current state
       * of the object.
       */
      struct StateException: virtual Exception, Utilities::StateException
      {
        StateException() {}
        StateException(string l): Utilities::Exception(l) {}
        StateException(string l, string m): Utilities::Exception(l, m) {}
      };

      /**
       * Internal error encountered
       */
      struct InternalException: virtual Exception, Utilities::
          InternalException
      {
        InternalException() {}
        InternalException(string l): Utilities::Exception(l) {}
        InternalException(string l, string m): Utilities::Exception(l, m) {}
      };
    }
  }
}

#endif // ARCHON_X3D_PROXY_EXCEPTION_H
```

Listing: **archon/x3d/proxy/exception.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#include <iostream>

#include <omniORB4/CORBA.h>
```



```
#include <archon/util/text.H>

#include <archon/x3d/proxy/exception.H>
#include <archon/x3d/proxy/sai.H>

namespace Archon
{
  namespace X3D
  {
    namespace Proxy
    {
      using namespace std;
      using namespace Utilities;

      bool Exception::catchInfo(Formatter format, string &result)
      {
        try { throw; }
        catch(x3d::sai::ServerErrorException &e)
        {
          result = (*format)(typeid(e), "", "'" + string(e.message) + "'");
          return true;
        }
        catch(x3d::sai::ApplicationErrorException &e)
        {
          result = (*format)(typeid(e), "", string(e.message));
          return true;
        }
        catch(x3d::sai::FileRequestException &e)
        {
          result = (*format)(typeid(e), "", string(e.message));
          return true;
        }
        catch(x3d::sai::StreamReadException &e)
        {
          result = (*format)(typeid(e), "", string(e.message));
          return true;
        }
        catch(omniORB::fatalException &e)
        {
          result = (*format)(typeid(e), string(e.file()) + ":" +
                   Text::toString(e.line()), e.errmsg());
          return true;
        }
        catch(CORBA::Exception &e)
        {
          result = (*format)(typeid(e), "", "");
          return true;
          abort();
        }
        catch(...)
        {
          return Utilities::Exception::catchInfo(format, result);
        }
      }

      void Exception::terminal()
      {
        string message;
```

```
        if(!catchInfo(&Exception::explainLong, message))
          message = explainLongUnknown();
        cerr << message;
        abort();
      }

      string Exception::catchShortExplanation()
      {
        string message;
        if(!catchInfo(&Exception::explainShort, message))
          message = explainShortUnknown();
        return message;
      }
    }
  }
}
```

Listing: **archon/x3d/proxy/field.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_PROXY_FIELD_H
#define ARCHON_X3D_PROXY_FIELD_H

#include <archon/x3d/proxy/node.H>
#include <archon/x3d/proxy/event.H>

namespace Archon
{
  namespace X3D
  {
    namespace Proxy
    {
      template<typename F> struct EventSource;
      template<typename F> struct EventTarget;

      struct Field
      {
      protected:
        friend struct ExecutionContext;
```



```
        Field() {} // Dummy
        Field(NodeBase *s, unsigned long i): self(s), id(i) {}

        NodeBase *self;
        unsigned long id;

        template<typename T> static void checkDerivation(T *) {}

        template<typename F, typename G>
        static void registerInterest(EventSource<F> *f,
                                     EventHandler<G> *h, bool destroy)
        {
          if(false) { F *p; checkDerivation<G>(p); }
          f->self->registerFieldInterest(f->id, h, destroy);
        }

        template<typename C, typename D>
        static void registerInterest(EventSource<SFNode<C> > *f,
                                     EventHandler<SFNode<C> > *h, bool
                                          destroy)
        {
          if(false) { C *p; checkDerivation<D>(p); }
          f->self->registerFieldInterest(f->id, h, destroy);
        }

        template<typename C, typename D>
        static void registerInterest(EventSource<MFNode<C> > *f,
                                     EventHandler<MFNode<C> > *h, bool
                                          destroy)
        {
          if(false) { C *p; checkDerivation<D>(p); }
          f->self->registerFieldInterest(f->id, h, destroy);
        }
      };

      template<typename F> struct FieldGet: virtual Field
      {
        typedef typename F::Type T;
        T get() const
        {
          x3d::sai::Value_var w = self->fieldGet(id);
          T v;
          self->importValue(F::index, w,
                            reinterpret_cast<void *>(&v));
          return v;
        }

      protected:
        FieldGet() {}
      };

      template<typename C> struct FieldGet<SFNode<C> >: virtual Field
      {
        typedef typename SFNode<C>::Type T;
        T get() const
        {
          x3d::sai::Value_var w = self->fieldGet(id);
          Ref<NodeBase> v;
```

```
    self->importValue(SFNode<C>::index, w,
                      reinterpret_cast<void *>(&v));
    C *c = dynamic_cast<C *>(v.get());
    if(!c) ARCHON_THROW1(Session::ProtocolException,
                         "Incompatible node type");
    return c;
  }

protected:
  FieldGet() {}
};

template<typename C> struct FieldGet<MFNode<C> >: virtual Field
{
  typedef typename MFNode<C>::Type T;
  T get() const
  {
    x3d::sai::Value_var w = self->fieldGet(id);
    vector<Ref<NodeBase> > v;
    self->importValue(MFNode<C>::index, w,
                      reinterpret_cast<void *>(&v));
    vector<Ref<C> > u(v.size());
    for(unsigned i=0; i<v.size(); ++i)
    {
      C *c = dynamic_cast<C *>(v[i].get());
      if(!c) ARCHON_THROW1(Session::ProtocolException,
                           "Incompatible node type");
      u[i].reset(c);
    }
    return u;
  }

protected:
  FieldGet() {}
};

template<typename F> struct FieldSet: virtual Field
{
  typedef typename F::ArgType A;
  void set(A v)
  {
    x3d::sai::Value w;
    self->exportValue(F::index,
                      reinterpret_cast<const void *>(&v), w);
    self->fieldSet(id, w);
  }

protected:
  FieldSet() {}
};

template<typename C> struct FieldSet<SFNode<C> >: virtual Field
{
  typedef typename SFNode<C>::ArgType A;
  void set(A v)
  {
    Ref<NodeBase> u = v;
    x3d::sai::Value w;
```



```
    self->exportValue(SFNode<C>::index,
                      reinterpret_cast<const void *>(&u), w);
    self->fieldSet(id, w);
  }

protected:
  FieldSet() {}
};

template<typename C> struct FieldSet<MFNode<C> >: virtual Field
{
  typedef typename MFNode<C>::ArgType A;
  void set(A v)
  {
    vector<Ref<NodeBase> > u(v.size());
    for(unsigned i=0; i<v.size(); ++i) u[i] = v[i];
    x3d::sai::Value w;
    self->exportValue(MFNode<C>::index,
                      reinterpret_cast<const void *>(&u), w);
    self->fieldSet(id, w);
  }

protected:
  FieldSet() {}
};

template<typename F> struct FieldSeqGet: virtual FieldGet<F>
{
  typedef typename F::Single SF;
  typedef typename SF::Type ST;

  ST getAt(unsigned long i) const
  {
    x3d::sai::Value_var w = self->fieldGetAt(id, i);
    ST v;
    self->importValue(SF::index, w,
                      reinterpret_cast<void *>(&v));
    return v;
  }

protected:
  FieldSeqGet() {}
};

template<typename C>
struct FieldSeqGet<MFNode<C> >: virtual FieldGet<MFNode<C> >
{
  typedef typename MFNode<C>::Single SF;
  typedef typename SF::Type ST;

  ST getAt(unsigned long i) const
  {
    x3d::sai::Value_var w = self->fieldGetAt(id, i);
    Ref<NodeBase> v;
    self->importValue(SF::index, w,
                      reinterpret_cast<void *>(&v));
    C *c = dynamic_cast<C *>(v.get());
    if(!c) ARCHON_THROW1(Session::ProtocolException,
```

```
                         "Incompatible node type");
    return c;
  }

protected:
  FieldSeqGet() {}
};

template<typename F> struct FieldSeqSet: virtual FieldSet<F>
{
  typedef typename F::Single SF;
  typedef typename SF::ArgType SA;

  void setAt(SA v, unsigned long i)
  {
    x3d::sai::Value w;
    self->exportValue(SF::index,
                      reinterpret_cast<const void *>(&v), w);
    self->fieldSetAt(id, w, i);
  }

  void add(SA v)
  {
    x3d::sai::Value w;
    self->exportValue(SF::index,
                      reinterpret_cast<const void *>(&v), w);
    self->fieldAdd(id, w);
  }

  void remove(SA v)
  {
    x3d::sai::Value w;
    self->exportValue(SF::index,
                      reinterpret_cast<const void *>(&v), w);
    self->fieldDel(id, w);
  }

protected:
  FieldSeqSet() {}
};

template<typename C>
struct FieldSeqSet<MFNode<C> >: virtual FieldSet<MFNode<C> >
{
  typedef typename MFNode<C>::Single SF;
  typedef typename SF::ArgType SA;

  void setAt(SA v, unsigned long i)
  {
    Ref<NodeBase> u = v;
    x3d::sai::Value w;
    self->exportValue(SF::index,
                      reinterpret_cast<const void *>(&u), w);
    self->fieldSetAt(id, w, i);
  }

  void add(SA v)
  {
```



```
    Ref<NodeBase> u = v;
    x3d::sai::Value w;
    self->exportValue(SF::index,
                      reinterpret_cast<const void *>(&u), w);
    self->fieldAdd(id, w);
  }

  void remove(SA v)
  {
    Ref<NodeBase> u = v;
    x3d::sai::Value w;
    self->exportValue(SF::index,
                      reinterpret_cast<const void *>(&u), w);
    self->fieldDel(id, w);
  }

protected:
  FieldSeqSet() {}
};



template<typename F>
struct EventSource: virtual FieldGet<F>
{
  template<typename G>
  void registerInterest(Ref<EventHandler<G> > h)
  {
    Field::registerInterest(this, h.get(), false);
  }

  template<typename G>
  void unregisterInterest(Ref<EventHandler<G> > h)
  {
    Field::registerInterest(this, h.get(), true);
  }
};

template<typename F>
struct EventTarget: virtual FieldSet<F>
{
};



template<typename F>
struct EventSourceSeq: FieldSeqGet<F>, EventSource<F> {};

template<typename F>
struct EventTargetSeq: FieldSeqSet<F>, EventTarget<F> {};



// Single value fields

template<typename F>
struct InitializeOnly: FieldGet<F>, FieldSet<F>
{
```

```
  typedef typename F::Type T;
  typedef typename F::ArgType A;

  InitializeOnly &operator=(A a) { set(a); return *this; }
  operator T() const { return get(); }

  InitializeOnly(NodeBase *s, unsigned long i): Field(s, i) {}
};

template<typename F>
struct InputOnly: EventTarget<F>
{
  typedef typename F::ArgType A;

  InputOnly &operator=(A a) { set(a); return *this; }
  void operator()(A a) { set(a); }

  InputOnly(NodeBase *s, unsigned long i): Field(s, i) {}
};

template<typename F>
struct OutputOnly: EventSource<F>
{
  typedef typename F::Type T;

  operator T() const { return get(); }

  OutputOnly(NodeBase *s, unsigned long i): Field(s, i) {}
};

template<typename F>
struct InputOutput: EventSource<F>, EventTarget<F>
{
  typedef typename F::Type T;
  typedef typename F::ArgType A;

  InputOutput &operator=(A a) { set(a); return *this; }
  operator T() const { return get(); }

  InputOutput(NodeBase *s, unsigned long i): Field(s, i) {}
};



// Sequence fields

template<typename F>
struct InitializeOnlySeq: FieldSeqGet<F>, FieldSeqSet<F>
{
  typedef typename F::Type T;
  typedef typename F::ArgType A;

  InitializeOnlySeq &operator=(A a) { set(a); return *this; }
  operator T() const { return get(); }

  InitializeOnlySeq(NodeBase *s, unsigned long i): Field(s, i) {}
};
```



```
      template<typename F>
      struct InputOnlySeq: EventTargetSeq<F>
      {
        typedef typename F::ArgType A;

        InputOnlySeq &operator=(A a) { set(a); return *this; }
        void operator()(A a) { set(a); }

        InputOnlySeq(NodeBase *s, unsigned long i): Field(s, i) {}
      };

      template<typename F>
      struct OutputOnlySeq: EventSourceSeq<F>
      {
        typedef typename F::Type T;

        operator T() const { return get(); }

        OutputOnlySeq(NodeBase *s, unsigned long i): Field(s, i) {}
      };

      template<typename F>
      struct InputOutputSeq: EventSourceSeq<F>, EventTargetSeq<F>
      {
        typedef typename F::Type T;
        typedef typename F::ArgType A;

        InputOutputSeq &operator=(A a) { set(a); return *this; }
        operator T() const { return get(); }

        InputOutputSeq(NodeBase *s, unsigned long i): Field(s, i) {}
      };
    }
  }
}

#endif // ARCHON_X3D_PROXY_FIELD_H
```

Listing: **archon/x3d/proxy/field_type.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */
```

```
#ifndef ARCHON_X3D_PROXY_FIELD_TYPE_H
#define ARCHON_X3D_PROXY_FIELD_TYPE_H

/**
 * \file field_type.H
 *
 * X3D field types
 */

#include <string>
#include <vector>

#include <archon/math/vector.H>
#include <archon/math/geometry.H>

#include <archon/util/ref.H>
#include <archon/util/time.H>
#include <archon/util/image.H>

namespace Archon
{
  namespace X3D
  {
    namespace Proxy
    {
      using namespace std;
      using namespace Math;
      using namespace Utilities;

      struct SFBool
      {
        typedef bool Type;
        typedef Type ArgType;
        static const unsigned index = 0;
      };

      struct MFBool
      {
        typedef SFBool Single;
        typedef vector<Single::Type> Type;
        typedef const Type &ArgType;
        static const unsigned index = 1;
      };


      struct SFColor
      {
        typedef double ElemType;
        static const unsigned size = 3;
        typedef BasicVector<ElemType, size> Type;
        typedef Type ArgType;
        static const unsigned index = 2;
      };

      struct MFColor
      {
        typedef SFColor Single;
```



```
        typedef vector<Single::Type> Type;
        typedef const Type &ArgType;
        static const unsigned index = 3;
      };


      struct SFColorRGBA
      {
        typedef double ElemType;
        static const unsigned size = 4;
        typedef BasicVector<ElemType, size> Type;
        typedef Type ArgType;
        static const unsigned index = 4;
      };

      struct MFColorRGBA
      {
        typedef SFColorRGBA Single;
        typedef vector<Single::Type> Type;
        typedef const Type &ArgType;
        static const unsigned index = 5;
      };


      struct SFDouble
      {
        typedef double Type;
        typedef Type ArgType;
        static const unsigned index = 6;
      };

      struct MFDouble
      {
        typedef SFDouble Single;
        typedef vector<Single::Type> Type;
        typedef const Type &ArgType;
        static const unsigned index = 7;
      };


      struct SFFloat
      {
        typedef double Type;
        typedef Type ArgType;
        static const unsigned index = 8;
      };

      struct MFFloat
      {
        typedef SFFloat Single;
        typedef vector<Single::Type> Type;
        typedef const Type &ArgType;
        static const unsigned index = 9;
      };


      struct SFImage
      {
```

```
  typedef Image Type;
  typedef const Type &ArgType;
  static const unsigned index = 10;
};

struct MFImage
{
  typedef SFImage Single;
  typedef vector<Single::Type> Type;
  typedef const Type &ArgType;
  static const unsigned index = 11;
};


struct SFInt32
{
  typedef int Type;
  typedef Type ArgType;
  static const unsigned index = 12;
};

struct MFInt32
{
  typedef SFInt32 Single;
  typedef vector<Single::Type> Type;
  typedef const Type &ArgType;
  static const unsigned index = 13;
};


template<typename C> struct SFNode
{
  typedef Ref<C> Type;
  typedef Type ArgType;
  static const unsigned index = 14;
};

template<typename C> struct MFNode
{
  typedef SFNode<C> Single;
  typedef vector<typename Single::Type> Type;
  typedef const Type &ArgType;
  static const unsigned index = 15;
};


struct SFRotation
{
  typedef double ElemType;
  static const unsigned size = 4;
  typedef AxisRot3 Type;
  typedef Type ArgType;
  static const unsigned index = 16;
};

struct MFRotation
{
  typedef SFRotation Single;
```



```
  typedef vector<Single::Type> Type;
  typedef const Type &ArgType;
  static const unsigned index = 17;
};


struct SFString
{
  typedef string Type;
  typedef Type ArgType;
  static const unsigned index = 18;
};

struct MFString
{
  typedef SFString Single;
  typedef vector<Single::Type> Type;
  typedef const Type &ArgType;
  static const unsigned index = 19;
};


struct SFTime
{
  typedef long ElemType;
  static const unsigned size = 2;
  typedef Time Type;
  typedef Type ArgType;
  static const unsigned index = 20;
};

struct MFTime
{
  typedef SFTime Single;
  typedef vector<Single::Type> Type;
  typedef const Type &ArgType;
  static const unsigned index = 21;
};


struct SFVec2d
{
  typedef SFDouble::Type ElemType;
  static const unsigned size = 2;
  typedef BasicVector<ElemType, size> Type;
  typedef Type ArgType;
  static const unsigned index = 22;
};

struct MFVec2d
{
  typedef SFVec2d Single;
  typedef vector<Single::Type> Type;
  typedef const Type &ArgType;
  static const unsigned index = 23;
};
```

```
      struct SFVec2f
      {
        typedef SFFloat::Type ElemType;
        static const unsigned size = 2;
        typedef BasicVector<ElemType, size> Type;
        typedef Type ArgType;
        static const unsigned index = 24;
      };

      struct MFVec2f
      {
        typedef SFVec2f Single;
        typedef vector<Single::Type> Type;
        typedef const Type &ArgType;
        static const unsigned index = 25;
      };


      struct SFVec3d
      {
        typedef SFDouble::Type ElemType;
        static const unsigned size = 3;
        typedef BasicVector<ElemType, size> Type;
        typedef Type ArgType;
        static const unsigned index = 26;
      };

      struct MFVec3d
      {
        typedef SFVec3d Single;
        typedef vector<Single::Type> Type;
        typedef const Type &ArgType;
        static const unsigned index = 27;
      };


      struct SFVec3f
      {
        typedef SFFloat::Type ElemType;
        static const unsigned size = 3;
        typedef BasicVector<ElemType, size> Type;
        typedef Type ArgType;
        static const unsigned index = 28;
      };

      struct MFVec3f
      {
        typedef SFVec3f Single;
        typedef vector<Single::Type> Type;
        typedef const Type &ArgType;
        static const unsigned index = 29;
      };

      const unsigned fieldTypeIndexRoof = 30;
    }
  }
}
```



```
#endif // ARCHON_X3D_PROXY_FIELD_TYPE_H
```

Listing: **archon/x3d/proxy/node.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_PROXY_NODE_H
#define ARCHON_X3D_PROXY_NODE_H

#include <archon/x3d/proxy/context.H>

namespace Archon
{
  namespace X3D
  {
    namespace Proxy
    {
      struct Session;
      struct ExecutionContext;

      /**
       * Virtual base class for all proxy X3D node types.
       */
      struct NodeBase: virtual RefObjectBase
      {
        Ref<Session> getSession() const { return session; }

      protected:
        friend struct Session;
        friend struct ExecutionContext;
        friend struct Field;
        template<typename F> friend struct FieldGet;
        template<typename F> friend struct FieldSeqGet;
        template<typename F> friend struct FieldSet;
        template<typename F> friend struct FieldSeqSet;

        NodeBase() {} // Dummy

        NodeBase(Session *s, unsigned long id):
          session(s), id(id), transCount(0) {}
```

```
        ~NodeBase() {{ session->destroyNode(this); }}

        template<typename T>
        static Ref<T> create(ExecutionContext *c,
                             unsigned long nodeTypeIndex)
        {
          Ref<NodeBase> n = c->session->createNode(c, nodeTypeIndex);
          return dynamic_cast<T *>(n.get());
        }

        const Ref<Session> session;

      private:
        unsigned long id;
        unsigned long transCount;

        void importValue(unsigned long id,
                         x3d::sai::Value &w, void *v) const
        {
          session->im(id, w, v);
        }

        void exportValue(unsigned long id,
                         const void *v, x3d::sai::Value &w) const
        {
          session->ex(id, v, w);
        }

        x3d::sai::Value *fieldGet(unsigned long id);
        x3d::sai::Value *fieldGetAt(unsigned long id, unsigned long index);

        void fieldSet(unsigned long id, x3d::sai::Value &);
        void fieldSetAt(unsigned long id, x3d::sai::Value &,
                        unsigned long index);
        void fieldAdd(unsigned long id, x3d::sai::Value &);
        void fieldDel(unsigned long id, x3d::sai::Value &);

        void registerFieldInterest(unsigned long fi,
                                   EventHandlerBase *h, bool destroy)
        {
          session->registerFieldInterest(id, fi, h, destroy);
        }
      };
    }
  }
}

#endif // ARCHON_X3D_PROXY_NODE_H
```

Listing: **archon/x3d/proxy/node.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
```



```
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#include <archon/x3d/proxy/node.H>

namespace Archon
{
  namespace X3D
  {
    namespace Proxy
    {
      x3d::sai::Value *NodeBase::fieldGet(unsigned long fi)
      {
        return session->session->getFieldValue(id, fi);
      }

      x3d::sai::Value *NodeBase::fieldGetAt(unsigned long fi, unsigned long i
          )
      {
        return session->session->getFieldValueAt(id, fi, i);
      }

      void NodeBase::fieldSet(unsigned long fi, x3d::sai::Value &v)
      {
        session->session->setFieldValue(id, fi, v);
      }

      void NodeBase::fieldSetAt(unsigned long fi, x3d::sai::Value &v,
                                unsigned long i)
      {
        session->session->setFieldValueAt(id, fi, i, v);
      }

      void NodeBase::fieldAdd(unsigned long fi, x3d::sai::Value &v)
      {
        session->session->setFieldValueAdd(id, fi, v);
      }

      void NodeBase::fieldDel(unsigned long fi, x3d::sai::Value &v)
      {
        session->session->setFieldValueDel(id, fi, v);
      }
    }
  }
}
```

Listing: **archon/x3d/proxy/orb.H**

```
/*
```

```
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */
#ifndef ARCHON_X3D_PROXY_ORB_H
#define ARCHON_X3D_PROXY_ORB_H

#include <omniORB4/CORBA.h>

#include <archon/util/thread.H>


namespace Archon
{
  namespace X3D
  {
    namespace Proxy
    {
      using namespace Utilities;

      struct Orb
      {
        Orb(int port=-1);
        ~Orb();

        CORBA::ORB_var &getCorbaOrb()     { return orb; }
        PortableServer::POA_var &getPoa() { return poa; }

      private:
        CORBA::ORB_var orb;
        PortableServer::POA_var poa;
      };

      template<typename T>
      struct ServantRef
      {
        ServantRef(T *p): p(p) {}
        ~ServantRef() {{ p->_remove_ref(); }}
        T *operator->() { return p; }
        T *get() const { return p; }
      private:
        T *p;
      };
    }
  }
```



```
}

#endif // ARCHON_X3D_PROXY_ORB_H
```

Listing: **archon/x3d/proxy/orb.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <iostream>

#include <archon/util/logger.H>
#include <archon/util/text.H>

#include <archon/x3d/proxy/orb.H>

namespace Archon
{
  namespace X3D
  {
    namespace Proxy
    {
      using namespace Utilities;

      Orb::Orb(int port)
      {
        char hostname[256];
        if(gethostname(hostname, sizeof(hostname))<0)
        {
          Logger::get()->log((string)strerror(errno) + "\n");
          abort();
        };

        hostname[sizeof(hostname)-1] = '\0';

        char endpoint[512];
        if(port<0) snprintf(endpoint, sizeof(endpoint), "giop:tcp:%s:",
             hostname);
```

```
        else snprintf(endpoint, sizeof(endpoint), "giop:tcp:%s:%d", hostname
             , port);

        const char *args[][2] =
          {
            { "endPoint"                , endpoint  },
          /*{ "offerBiDirectionalGIOP" , "1"       },
            { "acceptBiDirectionalGIOP", "1"       },
            { "clientTransportRule"    , "* tcp"   },
            { "serverTransportRule"    , "* tcp"   },*/
            { 0                        , 0         }
          };

        // Initialise the ORB.
        int argc = 0;
        orb = CORBA::ORB_init(argc, 0, "omniORB4", args);

        // Obtain a reference to the root POA.
        CORBA::Object_var obj = orb->resolve_initial_references("RootPOA");
        poa = PortableServer::POA::_narrow(obj);

        PortableServer::POA_var rootPoa = PortableServer::POA::_narrow(obj);

        CORBA::PolicyList policies(1);
        policies.length(1);
        policies[0] = new BiDirPolicy::BidirectionalPolicy(BiDirPolicy::BOTH
             );
        poa = rootPoa->create_POA("archon", PortableServer::POAManager::_nil
             (), policies);

        // Obtain a POAManager, and tell the POA to start accepting
        // requests on its objects.
        PortableServer::POAManager_var pman = poa->the_POAManager();
        pman->activate();
      }

      Orb::~Orb()
      {{ // The extra scope is needed to work around gcc3.2 bug #8287
        orb->destroy();
      }}
    }
  }
}
```

Listing: **archon/x3d/proxy/proxy.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
```



```
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

/**
 * \file proxy.H
 *
 * This header file that you should include in your X3D/SAI/Proxy
 * applications.
 */

#ifndef ARCHON_X3D_PROXY_PROXY_H
#define ARCHON_X3D_PROXY_PROXY_H

#include <archon/x3d/proxy/node.autogen.H>
#include <archon/x3d/proxy/application.H>

#endif // ARCHON_X3D_PROXY_PROXY_H
```

Listing: **archon/x3d/proxy/session.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_PROXY_SESSION_H
#define ARCHON_X3D_PROXY_SESSION_H

#include <archon/util/logger.H>
#include <archon/util/ref.H>
#include <archon/util/uri.H>

#include <archon/x3d/proxy/config.autogen.H>
#include <archon/x3d/proxy/exception.H>
#include <archon/x3d/proxy/orb.H>
#include <archon/x3d/proxy/disposer.H>
#include <archon/x3d/proxy/type.H>
#include <archon/x3d/proxy/value.H>

namespace Archon
```

```
{
  namespace X3D
  {
    namespace Proxy
    {
      using namespace std;
      using namespace Utilities;

      struct Application;
      struct ExecutionContext;
      struct NodeBase;
      struct EventHandlerBase;

      struct Session: virtual BackRefObjectBase, private Porter
      {
        static Ref<Session> create(string serverName,
                                   int corbaEndpointPort = -1,
                                   Logger * = Logger::get());

        void beginUpdate();
        void endUpdate();

        ~Session();

        struct Exception: virtual Proxy::Exception {};
        struct ArgumentException: Exception, Proxy::ArgumentException
        {
          ArgumentException(string l, string m): Utilities::Exception(l, m
               ) {}
        };
        struct ResourceException: Exception, Proxy::ResourceException
        {
          ResourceException() {}
          ResourceException(string l, string m): Utilities::Exception(l, m
               ) {}
        };
        struct StateException: Exception, Proxy::StateException
        {
          StateException(string l, string m): Utilities::Exception(l, m) {}
        };
        struct InternalException: Exception, Proxy::InternalException
        {
          InternalException(string l, string m): Utilities::Exception(l, m
               ) {}
        };
        /**
         * Some kind of error while attempting to make a connection to
         * the X3D server.
         */
        struct ConnectionException: ResourceException
        {
          ConnectionException(string l, string m): Utilities::Exception(l, m
               ) {}
        };
        /**
         * The requested node type of field is not supported by the
         * server.
         */
```



```
        struct SupportException: ResourceException
        {
          SupportException(string l, string m): Utilities::Exception(l, m) {}
        };
        /**
         * Data from the server appears to be corrupt.
         */
        struct ProtocolException: ResourceException
        {
          ProtocolException(string l, string m): Utilities::Exception(l, m
               ) {}
        };

      private:
        Session(string serverName, int corbaEndpointPort, Logger *);

        void refForwardDestroy();

        Logger *logger;

        x3d::sai::Server_var server;
        x3d::sai::Session_var session;


        template<typename F> friend struct EventHandler;

        friend struct ExecutionContext;
        void destroyContext(ExecutionContext *);

        Mutex contextMapMutex;
        vector<ExecutionContext *> contextMap;
        Ref<ExecutionContext> id2context(unsigned long, bool isApplication);
        unsigned long context2id(Ref<ExecutionContext>);


        friend struct NodeBase;
        Ref<NodeBase> createNode(ExecutionContext *, unsigned long
             nodeTypeIndex);
        void destroyNode(NodeBase *);

        Mutex nodeMapMutex;
        vector<NodeBase *> nodeMap;
        Ref<NodeBase> id2node(unsigned long) throw(IllegalNodeIdException);
        unsigned long node2id(Ref<NodeBase>);

        void registerFieldInterest(unsigned long ni, unsigned long fi,
                                   EventHandlerBase *, bool destroy);

        friend struct Application;
        Ref<Application> createApplication(const Uri &baseUri);
        Ref<Application> loadApplication(const Uri &uri);


        NodeLink nodeLinks[numberOfNodeTypes];
        unsigned long *nodeTypeMap;

        unsigned long nodeTypeIdBits;
        unsigned long nodeTypeIdMask;
```

```
        unsigned long nodeTypeIdRoof; // 1 + the largets node type ID
                                      // received from the server
                                      // during config.

        Mutex eventHandlersMutex;
        vector<Ref<EventHandlerBase> > eventHandlers;

        Orb orb;

        bool active;
        void activate();

        struct Servant;
        PortableServer::ObjectId_var servantId;

        // Application services
        void handleEvent(const x3d::sai::EventSeq &);
        x3d::sai::StreamReader_ptr requestFile(const char *);
        void log(const char *);

        x3d::sai::Server_ptr getServerReference(string name);

        struct StreamReaderServant;

        Ref<Disposer> disposer;
      };
    }
  }
}

#endif // ARCHON_X3D_PROXY_SESSION_H
```

Listing: **archon/x3d/proxy/session.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#include <archon/util/cxx_demangle.H>

#include <archon/x3d/proxy/session.H>

#include <archon/x3d/proxy/node.H>
```



```
#include <archon/x3d/proxy/application.H>
#include <archon/x3d/proxy/event.H>

#define CORBA_ENTRY_D(x) try { session->x; } \
catch(CORBA::UserException &) { throw; } \
catch(ForwardDestroyedException &) {} \
catch(...) { string m = Proxy::Exception::catchShortExplanation(); \
throw x3d::sai::ApplicationErrorException(m.c_str()); }

#define CORBA_ENTRY_V(x) try { session->x; } \
catch(CORBA::UserException &) { throw; } \
catch(ForwardDestroyedException &) \
{ throw x3d::sai::InvalidApplicationException(); } \
catch(...) { string m = Proxy::Exception::catchShortExplanation(); \
throw x3d::sai::ApplicationErrorException(m.c_str()); }

#define CORBA_ENTRY_R(x) try { return session->x; } \
catch(CORBA::UserException &) { throw; } \
catch(ForwardDestroyedException &) \
{ throw x3d::sai::InvalidApplicationException(); } \
catch(...) { string m = Proxy::Exception::catchShortExplanation(); \
throw x3d::sai::ApplicationErrorException(m.c_str()); }

namespace Archon
{
  namespace X3D
  {
    namespace Proxy
    {
      struct Session::Servant:
        POA_x3d::sai::Application, PortableServer::RefCountServantBase
      {
        const BackRef<Session> session;

        Servant(Ref<Session> s): session(s) {}

        void handleEvent(const x3d::sai::EventSeq &s)
        {
          CORBA_ENTRY_V(handleEvent(s));
        }

        x3d::sai::StreamReader_ptr requestFile(const char *path)
        {
          CORBA_ENTRY_R(requestFile(path));
        }

        void log(const char *message)
        {
          CORBA_ENTRY_V(log(message));
        }
      };


      template<typename T>
      struct AutoArrayPtr
      {
        AutoArrayPtr(unsigned long n): p(new T[n]) {}
        ~AutoArrayPtr() {{ delete[] p; }}
```

```
  T *p;
};


Ref<Session> Session::create(string serverName,
                             int corbaEndpointPort,
                             Logger *l)
{
  Ref<Session> s = new Session(serverName, corbaEndpointPort, l);
  s->activate();
  return s;
}

Session::Session(string serverName, int corbaEndpointPort, Logger *l):
  logger(l), orb(corbaEndpointPort), active(false)
{
  // Attempt to contact the server
  server = getServerReference(serverName);

  nodeTypeIdBits = server->getNodeTypeBitsInId();
  nodeTypeIdMask = (1 << nodeTypeIdBits) - 1;
  nodeTypeIdRoof = 0;

  for(unsigned long i=0; i<numberOfNodeTypes; ++i)
  {
    const NodeDef &d = nodeDefs[i];
    NodeLink &l = nodeLinks[i];
    try
    {
      unsigned long ti = server->getNodeType(d.name);
      if(ti >= nodeTypeIdRoof) nodeTypeIdRoof = ti+1;
      l.id = ti + 1;
    }
    catch(x3d::sai::InvalidNameException &)
    {
      logger->log("Server does not support node type '" +
                  string(d.name) + "'");
      continue;
    }

    if(!d.numberOfFields) continue;

    l.fields = new FieldLink[d.numberOfFields];

    for(unsigned long j=0; j<d.numberOfFields; ++j)
    {
      try
      {
        l.fields[j].id = server->
          getImmediateField(l.id-1, d.fields[j].name) + 1;
      }
      catch(x3d::sai::InvalidNameException &)
      {
        logger->log("Server does not support field '" +
                    string(d.name) + "." +
                    string(d.fields[j].name) + "'");
        continue;
      }
```



```
        l.fields[j].id = 0;
      }
    }


    // Create a mapping from Node type IDs to locally known Node
    // type indices.
    if(nodeTypeIdRoof > 1024)
      ARCHON_THROW1(ProtocolException, "Suspiciously many node types: " +
                    Text::toString(nodeTypeIdRoof));
    AutoArrayPtr<unsigned long> a(nodeTypeIdRoof);
    nodeTypeMap = a.p;
    for(unsigned long i=0; i<nodeTypeIdRoof; ++i) nodeTypeMap[i] = 0;
    for(unsigned long i=0; i<numberOfNodeTypes; ++i)
    {
      unsigned long id = nodeLinks[i].id;
      if(!id) continue;
      nodeTypeMap[id-1] = i+1;
    }
    a.p = 0;
  }

  Session::~Session()
  {{
    delete[] nodeTypeMap;
  }}

  void Session::refForwardDestroy()
  {
    if(active)
    {
      orb.getPoa()->deactivate_object(servantId);
      disposer->terminate();
      disposer.reset();
    }
  }

  /**
   * This cannot be done in the costructor since it creates a
   * reference to the session in the servant.
   *
   * \todo There is a potential problem here. If the server
   * contacts our session servant before the call to
   * 'server->createSession(ref)' returns and the call makes this
   * Session object attempt to contact the session object on the
   * server, then it would find 'session == 0'. I belive this
   * currtently cannot happen. The same is true for the disposer.
   */
  void Session::activate()
  {
    // Create a session on the server
    ServantRef<Servant> servant = new Servant(this);
    servantId = orb.getPoa()->activate_object(servant.get());
    x3d::sai::Application_var ref = servant->_this();
    session = server->createSession(ref);
    active = true;
    server = 0; // Release the server reference
```

```
  // Startup the disposer thread
  disposer = new Disposer(session);
  Thread::start(disposer);
}



void Session::beginUpdate()
{
  session->beginUpdate();
}

void Session::endUpdate()
{
  session->endUpdate();
}



Ref<Application> Session::createApplication(const Uri &u)
{
  CORBA::Boolean localFileUri = true;
  Ref<ExecutionContext> c =
    id2context(session->createApplicationScene(u.toString().c_str(),
                                               localFileUri), true);
  return dynamic_cast<Application *>(c.get());
}

Ref<Application> Session::loadApplication(const Uri &u)
{
  CORBA::Boolean localFileUri = true;
  Ref<ExecutionContext> c =
    id2context(session->loadApplicationScene(u.toString().c_str(),
                                             localFileUri), true);
  return dynamic_cast<Application *>(c.get());
}

void Session::destroyContext(ExecutionContext *c)
{
  unsigned long ci = c->id;
  {
    Mutex::Lock l(contextMapMutex);
    ExecutionContext *&m = contextMap[ci];
    if(m == c) m = 0;
  }

  disposer->addContext(c->id, c->transCount);
}

Ref<ExecutionContext> Session::id2context(unsigned long id, bool
     isApplication)
{
  if(!id) return 0;
  if(id > 1024L)
    ARCHON_THROW1(ProtocolException,
                  "Got suspiciously large context ID: " +
                  Text::toString(id));
```



```
  Mutex::Lock l(contextMapMutex);
  if(contextMap.size() <= id) contextMap.resize(id+1, 0);
  ExecutionContext *&m = contextMap[id];
  Ref<ExecutionContext> c(m, RefSafeIncTag());
  if(!c) c = isApplication ? new Application(this, id) :
    new ExecutionContext(this, id);
  ++c->transCount;
  m = c.get();
  return c;
}

unsigned long Session::context2id(Ref<ExecutionContext> c)
{
  return c->id;
}



Ref<NodeBase> Session::createNode(ExecutionContext *c, unsigned long i)
{
  unsigned long ti = nodeLinks[i].id;
  if(!ti)
  {
    ARCHON_THROW1(SupportException,
                  "Node type '" + string(nodeDefs[i].name) +
                  "' is not supported by the server");
    return 0;
  }
  --ti;
  return id2node(session->createNodeFast(c->id, ti));
}

void Session::destroyNode(NodeBase *n)
{
  unsigned long ni = n->id >> nodeTypeIdBits;
  {
    Mutex::Lock l(nodeMapMutex);
    NodeBase *&m = nodeMap[ni];
    if(m == n) m = 0;
  }

  disposer->addNode(n->id, n->transCount);
}

Ref<NodeBase> Session::id2node(unsigned long id)
  throw(IllegalNodeIdException)
{
  if(!id) return 0;
  unsigned long i = id >> nodeTypeIdBits;
  if(i > 1048576UL)
    ARCHON_THROW1(ProtocolException,
                  "Got suspiciously large node index: " +
                  Text::toString(i));
  Mutex::Lock l(nodeMapMutex);
  if(nodeMap.size() <= i) nodeMap.resize(i+1, 0);
  NodeBase *&m = nodeMap[i];
  Ref<NodeBase> n(m, RefSafeIncTag());
```

```
  if(!n)
  {
    unsigned long ti = id & nodeTypeIdMask;
    if(ti >= nodeTypeIdRoof)
      ARCHON_THROW1(ProtocolException,
                    "Got invalid node type ID from server: " +
                    Text::toString(ti));
    unsigned long i = nodeTypeMap[ti];
    if(!i) ARCHON_THROW1(ProtocolException,
                         "Got invalid node type ID from server: " +
                         Text::toString(ti));
    NodeInstantiator p = nodeDefs[i-1].instantiator;
    if(!p) ARCHON_THROW1(ProtocolException,
                         "Got abstract node type ID from server: " +
                         Text::toString(ti));
    n = (*p)(this, id);
  }
  ++n->transCount;
  m = n.get();
  return n;
}

unsigned long Session::node2id(Ref<NodeBase> n)
{
  return n->id;
}

/**
 * \todo Handle destruction
 */
void Session::registerFieldInterest(unsigned long ni, unsigned long fi,
                                    EventHandlerBase *h, bool destroy)
{
  unsigned long cookie = 0;
  {
    Mutex::Lock l(eventHandlersMutex);
    while(cookie < eventHandlers.size() && eventHandlers[cookie])
      ++cookie;
    if(cookie == eventHandlers.size()) eventHandlers.resize(cookie+1);
    eventHandlers[cookie].reset(h);
  }
  session->registerFieldInterest(ni, fi, cookie);
}


struct Session::StreamReaderServant:
  POA_x3d::sai::StreamReader, PortableServer::RefCountServantBase
{
  StreamReaderServant(Ref<Session> s, Ref<Stream::Reader> r):
    session(s), reader(r) {}

  virtual ~StreamReaderServant()
  {{ // The extra scope is needed to work around gcc3.2 bug #8287
  }}

  x3d::sai::chararray *read(CORBA::Long n)
  {
    AutoArrayPtr<char> buffer(n);
```



```
    try
    {
      int m = reader->readAll(buffer.p, n);
      x3d::sai::chararray_var a(new x3d::sai::chararray(m));
      a->length(m);
      for(int i=0; i<m; ++i) a[i] = buffer.p[i];
      return a._retn();
    }
    catch(Archon::Utilities::Stream::ReadException &e)
    {
      string m = e.getMessage();
      throw x3d::sai::StreamReadException(m.c_str());
    }
    catch(Archon::Utilities::ThreadTerminatedException &)
    {
      throw x3d::sai::StreamReadException("Reader was terminated");
    }
    catch(...)
    {
      string m = Proxy::Exception::catchShortExplanation();
      throw x3d::sai::StreamReadException(m.c_str());
    }
  }

  void dispose()
  {
    session->orb.getPoa()->deactivate_object(servantId);
  }

  Ref<Session> session;
  Ref<Stream::Reader> reader;
  PortableServer::ObjectId_var servantId;
};


void Session::handleEvent(const x3d::sai::EventSeq &s)
{
  for(unsigned i=0; i<s.length(); ++i)
  {
    const x3d::sai::Event &e = s[i];
    if(e.cookie < eventHandlers.size())
    {
      Mutex::Lock l(eventHandlersMutex);
      Ref<EventHandlerBase> h = eventHandlers[e.cookie];
      if(h)
      {
        h->handle(this, e.val);
        continue;
      }
    }
    ARCHON_THROW1(InternalException, "Illegal event handler index");
  }
}

x3d::sai::StreamReader_ptr Session::requestFile(const char *p)
{
  logger->log("Request for file \"" + string(p) + "\"");
  try
```

```
  {
    Ref<Stream::Reader> r = Stream::makeFileReader(p);
    ServantRef<StreamReaderServant> servant =
      new StreamReaderServant(this, r);
    servant->servantId = orb.getPoa()->activate_object(servant.get());
    x3d::sai::StreamReader_var ref = servant->_this();
    return ref._retn();
  }
  catch(Stream::FileOpenException &e)
  {
    string m = e.getMessage();
    throw x3d::sai::FileRequestException(m.c_str());
  }
}

void Session::log(const char *m)
{
  logger->log(m);
}



x3d::sai::Server_ptr Session::getServerReference(string name)
{
  CosNaming::NamingContext_var context;

  try
  {
    // Fetch the root naming context
    CORBA::Object_var obj =
      orb.getCorbaOrb()->resolve_initial_references("NameService");
    context = CosNaming::NamingContext::_narrow(obj);
    if(CORBA::is_nil(context))
      ARCHON_THROW1(ConnectionException,
                    "CORBA naming service is not available.");
  }
  catch(CORBA::Exception &)
  {
    ARCHON_THROW1(ConnectionException,
                  "CORBA naming service is not available.");
  }

  CORBA::Object_var obj;
  try
  {
    CosNaming::Name path;
    path.length(3);
    path[0].id = "x3d";
    path[1].id = "sai";
    path[2].id = name.c_str();
    obj = context->resolve(path);
  }
  catch(CosNaming::NamingContext::NotFound &)
  {
    ARCHON_THROW1(ConnectionException,
                  "Server '" + name + "' not found.");
  }
```



```
        x3d::sai::Server_var server;
        try
        {
          if(obj->_non_existent())
            ARCHON_THROW1(ConnectionException,
                          "Server '" + name + "' not found.");
          server = x3d::sai::Server::_narrow(obj);
        }
        catch(CORBA::Exception &)
        {
          ARCHON_THROW1(ConnectionException,
                        "Server '" + name + "' not found.");
        }
        if(CORBA::is_nil(server))
          ARCHON_THROW1(ConnectionException,
                        "Server '" + name + "' not found.");

        return server._retn();
      }
    }
  }
}
```

Listing: **archon/x3d/proxy/type.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_PROXY_TYPE_H
#define ARCHON_X3D_PROXY_TYPE_H

#include <string>

namespace Archon
{
  namespace X3D
  {
    namespace Proxy
    {
      using namespace std;

      struct NodeBase;
```

```
      struct Session;

      typedef NodeBase *(*NodeInstantiator)(Session *, unsigned long id);

      struct FieldDef
      {
        const char *name;
      };

      struct NodeDef
      {
        const char *name;
        NodeInstantiator instantiator;
        unsigned numberOfFields;
        const FieldDef *fields;
      };

      extern const NodeDef nodeDefs[];

      struct FieldLink
      {
        unsigned long id; // 1 + the ID known by the server, zero
                          // indicates "no server support"
      };

      struct NodeLink
      {
        unsigned long id; // 1 + the ID known by the server, zero
                          // indicates "no server support"
        FieldLink *fields;

        NodeLink(): id(0), fields(0) {}
        ~NodeLink() {{ if(fields) delete[] fields; }}
      };
    }
  }
}

#endif // ARCHON_X3D_PROXY_TYPE_H
```

Listing: **archon/x3d/proxy/value.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
```



```
 * (http://www.doxygen.org).
 */
#ifndef ARCHON_X3D_PROXY_VALUE_H
#define ARCHON_X3D_PROXY_VALUE_H

#include <archon/x3d/proxy/field_type.H>
#include <archon/x3d/proxy/sai.H>

namespace Archon
{
  namespace X3D
  {
    namespace Proxy
    {
      using namespace Utilities;

      struct NodeBase;

      struct Porter
      {
        struct TypeMismatchException: ArgumentException
        {
          TypeMismatchException(string l): Utilities::Exception(l) {}
        };

        struct IllegalNodeIdException: ArgumentException
        {
          IllegalNodeIdException(string l): Utilities::Exception(l) {}
        };

        virtual unsigned long node2id(Ref<NodeBase>) = 0;

        virtual Ref<NodeBase> id2node(unsigned long)
          throw(IllegalNodeIdException) = 0;


        void im(unsigned long type,
                const x3d::sai::Value &source, void *target)
          throw(TypeMismatchException, IllegalNodeIdException)
        {
          if(type != source.type) ARCHON_THROW(TypeMismatchException);
          (this->*importers[type])(source, target);
        }

        void ex(unsigned long type,
                const void *source, x3d::sai::Value &target)
        {
          (this->*exporters[type])(source, target);
          target.type = type;
        }


      protected:
        Porter();
        virtual ~Porter() {}

      private:
        typedef void (Porter::*Importer)
```

```
          (const x3d::sai::Value &source, void *target);

        typedef void (Porter::*Exporter)
          (const void *source, x3d::sai::Value &target);


        vector<Importer> importers;
        vector<Exporter> exporters;


        template<typename F> void imScalar(const x3d::sai::Value &, void *);
        template<typename F> void imTuple(const x3d::sai::Value &, void *);
        template<typename F> void imScalarSeq(const x3d::sai::Value &, void
              *);
        template<typename F> void imTupleSeq(const x3d::sai::Value &, void
              *);

        void imSFImage(const x3d::sai::Value &, void *);
        void imMFImage(const x3d::sai::Value &, void *);

        void imSFRotation(const x3d::sai::Value &, void *);
        void imMFRotation(const x3d::sai::Value &, void *);

        void imSFTime(const x3d::sai::Value &, void *);
        void imMFTime(const x3d::sai::Value &, void *);


        template<typename F> void exScalar(const void *, x3d::sai::Value &);
        template<typename F> void exTuple(const void *, x3d::sai::Value &);
        template<typename F> void exScalarSeq(const void *, x3d::sai::Value
              &);
        template<typename F> void exTupleSeq(const void *, x3d::sai::Value
              &);

        void exSFImage(const void *, x3d::sai::Value &);
        void exMFImage(const void *, x3d::sai::Value &);

        void exSFRotation(const void *, x3d::sai::Value &);
        void exMFRotation(const void *, x3d::sai::Value &);

        void exSFTime(const void *, x3d::sai::Value &);
        void exMFTime(const void *, x3d::sai::Value &);
      };
    }
  }
}

#endif // ARCHON_X3D_PROXY_VALUE_H
```

Listing: **archon/x3d/proxy/value.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
```



```
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#include <archon/x3d/proxy/value.H>
#include <archon/x3d/proxy/node.H>

namespace Archon
{
  namespace X3D
  {
    namespace Proxy
    {
      template<typename T> struct CorbaVal {};

      template<> struct CorbaVal<bool>
      {
        typedef CORBA::Boolean Type;
        typedef x3d::sai::booleanarray Array;
        typedef x3d::sai::booleanarray_var ArrayVar;
        static bool im(Porter *, Type v) { return bool(v); }
        static Type ex(Porter *, bool v) { return Type(v); }
        static bool fromAny(const CORBA::Any &a, Type &v)
        { return a >>= CORBA::Any::to_boolean(v); }
        static void toAny(CORBA::Any &a, Type v)
        { a <<= CORBA::Any::from_boolean(v); }
      };

      template<> struct CorbaVal<int>
      {
        typedef CORBA::Long Type;
        typedef x3d::sai::longarray Array;
        typedef x3d::sai::longarray_var ArrayVar;
        typedef x3d::sai::longdoublearray DoubleArray;
        typedef x3d::sai::longdoublearray_var DoubleArrayVar;
        static int im(Porter *, Type v) { return int(v); }
        static Type ex(Porter *, int v) { return Type(v); }
        static bool fromAny(const CORBA::Any &a, Type &v) { return a >>= v; }
        static void toAny(CORBA::Any &a, Type v) { a <<= v; }
      };

      template<> struct CorbaVal<long>
      {
        typedef CORBA::Long Type;
        typedef x3d::sai::longarray Array;
        typedef x3d::sai::longarray_var ArrayVar;
        typedef x3d::sai::longdoublearray DoubleArray;
        typedef x3d::sai::longdoublearray_var DoubleArrayVar;
        static long im(Porter *, Type v) { return long(v); }
        static Type ex(Porter *, long v) { return Type(v); }
```

```
  static bool fromAny(const CORBA::Any &a, Type &v) { return a >>= v; }
  static void toAny(CORBA::Any &a, Type v) { a <<= v; }
};

template<> struct CorbaVal<float>
{
  typedef CORBA::Float Type;
  typedef x3d::sai::floatarray Array;
  typedef x3d::sai::floatarray_var ArrayVar;
  typedef x3d::sai::floatdoublearray DoubleArray;
  typedef x3d::sai::floatdoublearray_var DoubleArrayVar;
  static float im(Porter *, Type v) { return float(v); }
  static Type ex(Porter *, float v) { return Type(v); }
  static bool fromAny(const CORBA::Any &a, Type &v) { return a >>= v; }
  static void toAny(CORBA::Any &a, Type v) { a <<= v; }
};

template<> struct CorbaVal<double>
{
  typedef CORBA::Double Type;
  typedef x3d::sai::doublearray Array;
  typedef x3d::sai::doublearray_var ArrayVar;
  typedef x3d::sai::doubledoublearray DoubleArray;
  typedef x3d::sai::doubledoublearray_var DoubleArrayVar;
  static double im(Porter *, Type v) { return double(v); }
  static Type ex(Porter *, double v) { return Type(v); }
  static bool fromAny(const CORBA::Any &a, Type &v) { return a >>= v; }
  static void toAny(CORBA::Any &a, Type v) { a <<= v; }
};

template<> struct CorbaVal<string>
{
  typedef const char *Type;
  typedef x3d::sai::stringarray Array;
  typedef x3d::sai::stringarray_var ArrayVar;
  static string im(Porter *, Type v) { return string(v); }
  static Type ex(Porter *, string v)
  { return CORBA::string_dup(v.c_str()); }
  static bool fromAny(const CORBA::Any &a, Type &v) { return a >>= v; }
  static void toAny(CORBA::Any &a, Type v) { a <<= v; }
};

template<> struct CorbaVal<Ref<NodeBase> >
{
  typedef CORBA::ULong Type;
  typedef x3d::sai::ulongarray Array;
  typedef x3d::sai::ulongarray_var ArrayVar;
  static Ref<NodeBase> im(Porter *p, Type v)
  { return p->id2node(static_cast<unsigned long>(v)); }
  static Type ex(Porter *p, Ref<NodeBase> v)
  { return Type(p->node2id(v)); }
  static bool fromAny(const CORBA::Any &a, Type &v) { return a >>= v; }
  static void toAny(CORBA::Any &a, Type v) { a <<= v; }
};


template<typename F>
```



```
void Porter::imScalar(const x3d::sai::Value &s, void *t)
{
  typedef typename F::Type T;
  typedef CorbaVal<T> C;
  typename C::Type w;
  if(!C::fromAny(s.val, w)) ARCHON_THROW(TypeMismatchException);
  T &v = *reinterpret_cast<T *>(t);
  v = C::im(this, w);
}

template<typename F>
void Porter::imTuple(const x3d::sai::Value &s, void *t)
{
  typedef typename F::Type T;
  typedef typename F::ElemType E;
  typedef CorbaVal<E> C;
  const typename C::Array *w;
  if(!(s.val >>= w)) ARCHON_THROW(TypeMismatchException);
  if(w->length() != F::size) ARCHON_THROW(TypeMismatchException);
  T &v = *reinterpret_cast<T *>(t);
  for(unsigned i=0; i<F::size; ++i) v[i] = C::im(this, (*w)[i]);
}

template<typename F>
void Porter::imScalarSeq(const x3d::sai::Value &s, void *t)
{
  typedef typename F::Type T;
  typedef typename F::Single::Type E;
  typedef CorbaVal<E> C;
  const typename C::Array *w;
  if(!(s.val >>= w)) ARCHON_THROW(TypeMismatchException);
  T &v = *reinterpret_cast<T *>(t);
  v.resize(w->length());
  for(unsigned i=0; i<w->length(); ++i) v[i] = C::im(this, (*w)[i]);
}

template<typename F>
void Porter::imTupleSeq(const x3d::sai::Value &s, void *t)
{
  typedef typename F::Type T;
  typedef typename F::Single S;
  typedef typename S::ElemType E;
  typedef CorbaVal<E> C;
  const typename C::DoubleArray *w;
  if(!(s.val >>= w)) ARCHON_THROW(TypeMismatchException);
  T &v = *reinterpret_cast<T *>(t);
  v.resize(w->length());
  for(unsigned i=0; i<w->length(); ++i)
  {
    typename S::Type &u = v[i];
    const typename C::Array &t = (*w)[i];
    if(t.length() != S::size) ARCHON_THROW(TypeMismatchException);
    for(unsigned j=0; j<S::size; ++j) u[j] = C::im(this, t[j]);
  }
}

/**
 * \todo Needs to be implemented
```

```
 */
void Porter::imSFImage(const x3d::sai::Value &s, void *t)
{
  ARCHON_THROW1(InternalException, "imSFImage not yet implemented");
}

/**
 * \todo Needs to be implemented
 */
void Porter::imMFImage(const x3d::sai::Value &s, void *t)
{
  ARCHON_THROW1(InternalException, "imMFImage not yet implemented");
}

void Porter::imSFRotation(const x3d::sai::Value &s, void *t)
{
  typedef SFRotation F;
  typedef F::Type T;
  typedef F::ElemType E;
  typedef CorbaVal<E> C;
  const C::Array *w;
  if(!(s.val >>= w)) ARCHON_THROW(TypeMismatchException);
  if(w->length() != F::size) ARCHON_THROW(TypeMismatchException);
  T &v = *reinterpret_cast<T *>(t);
  v.normalizedAxis[0] = C::im(this, (*w)[0]);
  v.normalizedAxis[1] = C::im(this, (*w)[1]);
  v.normalizedAxis[2] = C::im(this, (*w)[2]);
  v.angle = C::im(this, (*w)[3]);
}

void Porter::imMFRotation(const x3d::sai::Value &s, void *t)
{
  typedef MFRotation F;
  typedef F::Type T;
  typedef F::Single S;
  typedef S::ElemType E;
  typedef CorbaVal<E> C;
  const C::DoubleArray *w;
  if(!(s.val >>= w)) ARCHON_THROW(TypeMismatchException);
  T &v = *reinterpret_cast<T *>(t);
  v.resize(w->length());
  for(unsigned i=0; i<w->length(); ++i)
  {
    S::Type &u = v[i];
    const C::Array &t = (*w)[i];
    if(t.length() != S::size) ARCHON_THROW(TypeMismatchException);
    u.normalizedAxis[0] = C::im(this, t[0]);
    u.normalizedAxis[1] = C::im(this, t[1]);
    u.normalizedAxis[2] = C::im(this, t[2]);
    u.angle = C::im(this, t[3]);
  }
}

void Porter::imSFTime(const x3d::sai::Value &s, void *t)
{
  typedef SFTime F;
  typedef F::Type T;
  typedef F::ElemType E;
```



```
  typedef CorbaVal<E> C;
  const C::Array *w;
  if(!(s.val >>= w)) ARCHON_THROW(TypeMismatchException);
  if(w->length() != F::size) ARCHON_THROW(TypeMismatchException);
  T &v = *reinterpret_cast<T *>(t);
  v.set(C::im(this, (*w)[0]), C::im(this, (*w)[1]));
}

void Porter::imMFTime(const x3d::sai::Value &s, void *t)
{
  typedef MFTime F;
  typedef F::Type T;
  typedef F::Single S;
  typedef S::ElemType E;
  typedef CorbaVal<E> C;
  const C::DoubleArray *w;
  if(!(s.val >>= w)) ARCHON_THROW(TypeMismatchException);
  T &v = *reinterpret_cast<T *>(t);
  v.resize(w->length());
  for(unsigned i=0; i<w->length(); ++i)
  {
    const C::Array &t = (*w)[i];
    if(t.length() != S::size) ARCHON_THROW(TypeMismatchException);
    v[i].set(C::im(this, t[0]), C::im(this, t[1]));
  }
}



template<typename F>
void Porter::exScalar(const void *s, x3d::sai::Value &t)
{
  typedef typename F::Type T;
  typedef CorbaVal<T> C;
  const T &v = *reinterpret_cast<const T *>(s);
  C::toAny(t.val, C::ex(this, v));
}

template<typename F>
void Porter::exTuple(const void *s, x3d::sai::Value &t)
{
  typedef typename F::Type T;
  typedef typename F::ElemType E;
  typedef CorbaVal<E> C;
  const T &v = *reinterpret_cast<const T *>(s);
  typename C::ArrayVar w(new typename C::Array(F::size));
  w->length(F::size);
  for(unsigned i=0; i<F::size; ++i) w[i] = C::ex(this, v[i]);
  t.val <<= w._retn();
}

template<typename F>
void Porter::exScalarSeq(const void *s, x3d::sai::Value &t)
{
  typedef typename F::Type T;
  typedef typename F::Single::Type E;
  typedef CorbaVal<E> C;
```

```
  const T &v = *reinterpret_cast<const T *>(s);
  typename C::ArrayVar w(new typename C::Array(v.size()));
  w->length(v.size());
  for(unsigned i=0; i<v.size(); ++i) w[i] = C::ex(this, v[i]);
  t.val <<= w._retn();
}

template<typename F>
void Porter::exTupleSeq(const void *s, x3d::sai::Value &t)
{
  typedef typename F::Type T;
  typedef typename F::Single S;
  typedef typename S::ElemType E;
  typedef CorbaVal<E> C;
  const T &v = *reinterpret_cast<const T *>(s);
  typename C::DoubleArrayVar
    w(new typename C::DoubleArray(v.size()));
  w->length(v.size());
  for(unsigned i=0; i<v.size(); ++i)
  {
    typename C::Array &u = w[i];
    u.length(S::size);
    const typename S::Type &x = v[i];
    for(unsigned j=0; j<S::size; ++j) u[j] = C::ex(this, x[j]);
  }
  t.val <<= w._retn();
}

/**
 * \todo Needs to be implemented
 */
void Porter::exSFImage(const void *s, x3d::sai::Value &t)
{
  ARCHON_THROW1(InternalException, "exSFImage not yet implemented");
}

/**
 * \todo Needs to be implemented
 */
void Porter::exMFImage(const void *s, x3d::sai::Value &t)
{
  ARCHON_THROW1(InternalException, "exMFImage not yet implemented");
}

void Porter::exSFRotation(const void *s, x3d::sai::Value &t)
{
  typedef SFRotation F;
  typedef F::Type T;
  typedef F::ElemType E;
  typedef CorbaVal<E> C;
  const T &v = *reinterpret_cast<const T *>(s);
  C::ArrayVar w(new C::Array(F::size));
  w->length(F::size);
  w[0] = C::ex(this, v.normalizedAxis[0]);
  w[1] = C::ex(this, v.normalizedAxis[1]);
  w[2] = C::ex(this, v.normalizedAxis[2]);
  w[3] = C::ex(this, v.angle);
  t.val <<= w._retn();
```



```
}

void Porter::exMFRotation(const void *s, x3d::sai::Value &t)
{
  typedef MFRotation F;
  typedef F::Type T;
  typedef F::Single S;
  typedef S::ElemType E;
  typedef CorbaVal<E> C;
  const T &v = *reinterpret_cast<const T *>(s);
  C::DoubleArrayVar w(new C::DoubleArray(v.size()));
  w->length(v.size());
  for(unsigned i=0; i<v.size(); ++i)
  {
    C::Array &u = w[i];
    u.length(S::size);
    const S::Type &x = v[i];
    u[0] = C::ex(this, x.normalizedAxis[0]);
    u[1] = C::ex(this, x.normalizedAxis[1]);
    u[2] = C::ex(this, x.normalizedAxis[2]);
    u[3] = C::ex(this, x.angle);
  }
  t.val <<= w._retn();
}

void Porter::exSFTime(const void *s, x3d::sai::Value &t)
{
  typedef SFTime F;
  typedef F::Type T;
  typedef F::ElemType E;
  typedef CorbaVal<E> C;
  const T &v = *reinterpret_cast<const T *>(s);
  C::ArrayVar w(new C::Array(F::size));
  w->length(F::size);
  long sec;
  long nsec;
  v.get(sec, nsec);
  w[0] = C::ex(this, sec);
  w[1] = C::ex(this, nsec);
  t.val <<= w._retn();
}

void Porter::exMFTime(const void *s, x3d::sai::Value &t)
{
  typedef MFTime F;
  typedef F::Type T;
  typedef F::Single S;
  typedef S::ElemType E;
  typedef CorbaVal<E> C;
  const T &v = *reinterpret_cast<const T *>(s);
  C::DoubleArrayVar w(new C::DoubleArray(v.size()));
  w->length(v.size());
  for(unsigned i=0; i<v.size(); ++i)
  {
    C::Array &u = w[i];
    u.length(S::size);
    long sec;
    long nsec;
```

```
    v[i].get(sec, nsec);
    u[0] = C::ex(this, sec);
    u[1] = C::ex(this, nsec);
  }
  t.val <<= w._retn();
}

Porter::Porter()
{
  importers.resize(fieldTypeIndexRoof);
  exporters.resize(fieldTypeIndexRoof);

  exporters[SFBool::index] = &Porter::exScalar<SFBool>;
  importers[SFBool::index] = &Porter::imScalar<SFBool>;
  exporters[MFBool::index] = &Porter::exScalarSeq<MFBool>;
  importers[MFBool::index] = &Porter::imScalarSeq<MFBool>;

  exporters[SFColor::index] = &Porter::exTuple<SFColor>;
  importers[SFColor::index] = &Porter::imTuple<SFColor>;
  exporters[MFColor::index] = &Porter::exTupleSeq<MFColor>;
  importers[MFColor::index] = &Porter::imTupleSeq<MFColor>;

  exporters[SFColorRGBA::index] = &Porter::exTuple<SFColorRGBA>;
  importers[SFColorRGBA::index] = &Porter::imTuple<SFColorRGBA>;
  exporters[MFColorRGBA::index] = &Porter::exTupleSeq<MFColorRGBA>;
  importers[MFColorRGBA::index] = &Porter::imTupleSeq<MFColorRGBA>;

  exporters[SFDouble::index] = &Porter::exScalar<SFDouble>;
  importers[SFDouble::index] = &Porter::imScalar<SFDouble>;
  exporters[MFDouble::index] = &Porter::exScalarSeq<MFDouble>;
  importers[MFDouble::index] = &Porter::imScalarSeq<MFDouble>;

  exporters[SFFloat::index] = &Porter::exScalar<SFFloat>;
  importers[SFFloat::index] = &Porter::imScalar<SFFloat>;
  exporters[MFFloat::index] = &Porter::exScalarSeq<MFFloat>;
  importers[MFFloat::index] = &Porter::imScalarSeq<MFFloat>;

  exporters[SFImage::index] = &Porter::exSFImage;
  importers[SFImage::index] = &Porter::imSFImage;
  exporters[MFImage::index] = &Porter::exMFImage;
  importers[MFImage::index] = &Porter::imMFImage;

  exporters[SFInt32::index] = &Porter::exScalar<SFInt32>;
  importers[SFInt32::index] = &Porter::imScalar<SFInt32>;
  exporters[MFInt32::index] = &Porter::exScalarSeq<MFInt32>;
  importers[MFInt32::index] = &Porter::imScalarSeq<MFInt32>;

  exporters[SFNode<NodeBase>::index] =
    &Porter::exScalar<SFNode<NodeBase> >;
  importers[SFNode<NodeBase>::index] =
    &Porter::imScalar<SFNode<NodeBase> >;
  exporters[MFNode<NodeBase>::index] =
    &Porter::exScalarSeq<MFNode<NodeBase> >;
  importers[MFNode<NodeBase>::index] =
    &Porter::imScalarSeq<MFNode<NodeBase> >;

  exporters[SFRotation::index] = &Porter::exSFRotation;
  importers[SFRotation::index] = &Porter::imSFRotation;
```



```
        exporters[MFRotation::index] = &Porter::exMFRotation;
        importers[MFRotation::index] = &Porter::imMFRotation;

        exporters[SFString::index] = &Porter::exScalar<SFString>;
        importers[SFString::index] = &Porter::imScalar<SFString>;
        exporters[MFString::index] = &Porter::exScalarSeq<MFString>;
        importers[MFString::index] = &Porter::imScalarSeq<MFString>;

        exporters[SFTime::index] = &Porter::exSFTime;
        importers[SFTime::index] = &Porter::imSFTime;
        exporters[MFTime::index] = &Porter::exMFTime;
        importers[MFTime::index] = &Porter::imMFTime;

        exporters[SFVec2d::index] = &Porter::exTuple<SFVec2d>;
        importers[SFVec2d::index] = &Porter::imTuple<SFVec2d>;
        exporters[MFVec2d::index] = &Porter::exTupleSeq<MFVec2d>;
        importers[MFVec2d::index] = &Porter::imTupleSeq<MFVec2d>;

        exporters[SFVec2f::index] = &Porter::exTuple<SFVec2f>;
        importers[SFVec2f::index] = &Porter::imTuple<SFVec2f>;
        exporters[MFVec2f::index] = &Porter::exTupleSeq<MFVec2f>;
        importers[MFVec2f::index] = &Porter::imTupleSeq<MFVec2f>;

        exporters[SFVec3d::index] = &Porter::exTuple<SFVec3d>;
        importers[SFVec3d::index] = &Porter::imTuple<SFVec3d>;
        exporters[MFVec3d::index] = &Porter::exTupleSeq<MFVec3d>;
        importers[MFVec3d::index] = &Porter::imTupleSeq<MFVec3d>;

        exporters[SFVec3f::index] = &Porter::exTuple<SFVec3f>;
        importers[SFVec3f::index] = &Porter::imTuple<SFVec3f>;
        exporters[MFVec3f::index] = &Porter::exTupleSeq<MFVec3f>;
        importers[MFVec3f::index] = &Porter::imTupleSeq<MFVec3f>;
      }
    }
  }
}
```

Listing: **archon/x3d/sai/change.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */
#ifndef ARCHON_X3D_SAI_CHANGE_H
```

```
#define ARCHON_X3D_SAI_CHANGE_H

#include <archon/x3d/server/event.H>

namespace Archon
{
  namespace X3D
  {
    namespace SAI
    {
      using namespace Utilities;

      struct ChangeBase: virtual RefObjectBase
      {
        virtual void apply(Time t) const = 0;
      };

      struct FieldSetChange: ChangeBase
      {
        FieldSetChange(Ref<Node> node, const FieldBase *field,
                       Ref<const ValueBase> value):
          node(node), field(field), value(value) {}

        Ref<Node> node;
        const FieldBase *field;
        Ref<const ValueBase> value;

        void apply(Time t) const
        {
          Event e(value, t);
          node->handleEvent(field, &e);
        }
      };

      struct FieldSetAtChange: ChangeBase
      {
        FieldSetAtChange(Ref<Node> node, const SequenceFieldBase *field,
                         Ref<const ValueBase> value, unsigned index):
          node(node), field(field), value(value), index(index) {}

        Ref<Node> node;
        const SequenceFieldBase *field;
        Ref<const ValueBase> value;
        unsigned index;

        void apply(Time t) const
        {
          node->setAt(field, value, index, t);
        }
      };

      struct FieldListChange: ChangeBase
      {
        FieldListChange(Ref<Node> node, const SequenceFieldBase *field,
                        Ref<const ValueBase> value, bool remove = false):
          node(node), field(field), value(value), remove(remove) {}

        Ref<Node> node;
```



```
        const SequenceFieldBase *field;
        Ref<const ValueBase> value;
        bool remove;

        void apply(Time t) const
        {
          if(remove) node->remove(field, value, t);
          else node->add(field, value, t);
        }
      };

      struct RouteChange: ChangeBase
      {
        RouteChange(Ref<Node> sourceNode, const FieldBase *sourceField,
                    Ref<Node> targetNode, const FieldBase *targetField,
                    bool remove = false):
          sourceNode(sourceNode), sourceField(sourceField),
          targetNode(targetNode), targetField(targetField),
          remove(remove) {}

        Ref<Node> sourceNode;
        const FieldBase *sourceField;
        Ref<Node> targetNode;
        const FieldBase *targetField;
        bool remove;

        void apply(Time) const
        {
          if(remove) Route::del(sourceNode, sourceField,
                                targetNode, targetField);
          else Route::add(sourceNode, sourceField,
                          targetNode, targetField);
        }
      };
    }
  }
}

#endif // ARCHON_X3D_SAI_CHANGE_H
```

Listing: **archon/x3d/sai/exception.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
```

```
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_SAI_EXCEPTION_H
#define ARCHON_X3D_SAI_EXCEPTION_H

#include <string>

#include <archon/util/exception.H>

namespace Archon
{
  namespace X3D
  {
    namespace SAI
    {
      using namespace std;

      struct Exception: virtual Utilities::Exception
      {
        /**
         * \sa Utilities::Exception::catchInfo
         */
        static bool catchInfo(Formatter format, string &result);

        /**
         * \sa Utilities::Exception::terminal
         */
        static void terminal();

        static string catchShortExplanation();
      };


      /**
       * A problem with one or more of the method arguments
       */
      struct ArgumentException: virtual Exception, Utilities::
          ArgumentException
      {
        ArgumentException() {}
        ArgumentException(string l): Utilities::Exception(l) {}
        ArgumentException(string l, string m): Utilities::Exception(l, m) {}
      };

      /**
       * A problem with getting access to external resources
       */
      struct ResourceException: virtual Exception, Utilities::
          ResourceException
      {
        ResourceException() {}
        ResourceException(string l): Utilities::Exception(l) {}
        ResourceException(string l, string m): Utilities::Exception(l, m) {}
      };

      /**
       * The request operation is not compatible with the current state
```



```
       * of the object.
       */
      struct StateException: virtual Exception, Utilities::StateException
      {
        StateException() {}
        StateException(string l): Utilities::Exception(l) {}
        StateException(string l, string m): Utilities::Exception(l, m) {}
      };

      /**
       * Internal error encountered
       */
      struct InternalException: virtual Exception, Utilities::
          InternalException
      {
        InternalException() {}
        InternalException(string l): Utilities::Exception(l) {}
        InternalException(string l, string m): Utilities::Exception(l, m) {}
      };
    }
  }
}

#endif // ARCHON_X3D_SAI_EXCEPTION_H
```

Listing: **archon/x3d/sai/exception.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#include <iostream>

#include <omniORB4/CORBA.h>

#include <archon/util/text.H>

#include <archon/x3d/server/exception.H>

#include <archon/x3d/sai/exception.H>
#include <archon/x3d/sai/sai.H>

namespace Archon
```

```
{
  namespace X3D
  {
    namespace SAI
    {
      using namespace std;
      using namespace Utilities;

      bool Exception::catchInfo(Formatter format, string &result)
      {
        try { throw; }
        catch(x3d::sai::ServerErrorException &e)
        {
          result = (*format)(typeid(e), "", "'" + string(e.message) + "'");
          return true;
        }
        catch(x3d::sai::ApplicationErrorException &e)
        {
          result = (*format)(typeid(e), "", string(e.message));
          return true;
        }
        catch(x3d::sai::FileRequestException &e)
        {
          result = (*format)(typeid(e), "", string(e.message));
          return true;
        }
        catch(x3d::sai::StreamReadException &e)
        {
          result = (*format)(typeid(e), "", string(e.message));
          return true;
        }
        catch(omniORB::fatalException &e)
        {
          result = (*format)(typeid(e), string(e.file()) + ":" +
                  Text::toString(e.line()), e.errmsg());
          return true;
        }
        catch(CORBA::Exception &e)
        {
          result = (*format)(typeid(e), "", "");
          return true;
          abort();
        }
        catch(...)
        {
          return
            Utilities::Exception::catchInfo(format, result) ||
            X3D::Exception::catchInfo(format, result);
        }
      }

      void Exception::terminal()
      {
        string message;
        if(!catchInfo(&Exception::explainLong, message))
          message = explainLongUnknown();
        cerr << message;
        abort();
```



```
      }

      string Exception::catchShortExplanation()
      {
        string message;
        if(!catchInfo(&Exception::explainShort, message))
          message = explainShortUnknown();
        return message;
      }
    }
  }
}
```

Listing: **archon/x3d/sai/listener.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */
#ifndef ARCHON_X3D_SAI_LISTENER_H
#define ARCHON_X3D_SAI_LISTENER_H

#include <archon/util/time.H>

#include <archon/x3d/server/server.H>

#include <archon/x3d/sai/exception.H>
#include <archon/x3d/sai/session.H>

namespace Archon
{
  namespace X3D
  {
    namespace SAI
    {
      using namespace Utilities;

      struct ServerNameException: ResourceException
      {
        ServerNameException(string l, string m): Utilities::Exception(l, m
            ) {}
      };
```

```
      struct Listener: Server::Listener
      {
        Listener(Ref<Server> server,
                 Time applicationHeartbeat,
                 int corbaEndpointPort);

        ~Listener();

        Orb &getOrb() { return orb; }

        void startServingClientRequests(Ref<GroupingNode>
             applicationContainer);

        void removeSession(Ref<Session>);

        const BackRef<Server> server;

      private:
        void refForwardDestroy();

        Mutex mutex;

        const Time applicationHeartbeat;
        Ref<GroupingNode> applicationContainer;

        Orb orb;

        struct Servant;
        PortableServer::ObjectId_var servantId;
        bool active;

        /**
         * Activate the Server CORBA servant.
         */
        void activate();

        /**
         * Deactivate the Server CORBA servant.
         */
        void deactivate();

        CORBA::ULong getNodeTypeBitsInId();
        CORBA::ULong getNodeType(const char *);
        CORBA::ULong getImmediateField(CORBA::ULong, const char *);
        x3d::sai::Session_ptr createSession(x3d::sai::Application_ptr);

        list<Ref<Session> > sessions;
      };
    }
  }
}

#endif // ARCHON_X3D_SAI_LISTENER_H
```

Listing: **archon/x3d/sai/listener.C**

```
/*
 * This file is part of the "Archon" framework.
```



```
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#include <archon/util/cxx_demangle.H>
#include <archon/util/mutex.H>

#include <archon/x3d/sai/listener.H>

#define CORBA_ENTRY_D(x) try { listener->x; } \
catch(CORBA::UserException &) { throw; } \
catch(ForwardDestroyedException &) {} \
catch(...) { string m = SAI::Exception::catchShortExplanation(); \
throw x3d::sai::ServerErrorException(m.c_str()); }

#define CORBA_ENTRY_V(x) try { listener->x; } \
catch(CORBA::UserException &) { throw; } \
catch(ForwardDestroyedException &) \
{ throw x3d::sai::InvalidSessionException(); } \
catch(...) { string m = SAI::Exception::catchShortExplanation(); \
throw x3d::sai::ServerErrorException(m.c_str()); }

#define CORBA_ENTRY_R(x) try { return listener->x; } \
catch(CORBA::UserException &) { throw; } \
catch(ForwardDestroyedException &) \
{ throw x3d::sai::InvalidSessionException(); } \
catch(...) { string m = SAI::Exception::catchShortExplanation(); \
throw x3d::sai::ServerErrorException(m.c_str()); }

namespace Archon
{
  namespace X3D
  {
    namespace SAI
    {
      struct Listener::Servant:
        POA_x3d::sai::Server, PortableServer::RefCountServantBase
      {
        const BackRef<Listener> listener;

        Servant(Ref<Listener> listener): listener(listener) {}

        CORBA::ULong getNodeTypeBitsInId()
        {
          CORBA_ENTRY_R(getNodeTypeBitsInId());
```

```
  }

  CORBA::ULong getNodeType(const char *typeName)
  {
    CORBA_ENTRY_R(getNodeType(typeName));
  }

  CORBA::ULong getImmediateField(CORBA::ULong nodeTypeId,
                                 const char *fieldName)
  {
    CORBA_ENTRY_R(getImmediateField(nodeTypeId, fieldName));
  }

  x3d::sai::Session_ptr createSession(x3d::sai::Application_ptr a)
  {
    CORBA_ENTRY_R(createSession(a));
  }
};


Listener::Listener(Ref<Server> server,
                   Time applicationHeartbeat,
                   int corbaEndpointPort):
  server(server),
  applicationHeartbeat(applicationHeartbeat),
  orb(corbaEndpointPort), active(false)
{
}

Listener::~Listener()
{{
}}

void Listener::activate()
{
  if(active) return;
  ServantRef<Servant> servant = new Servant(this);
  servantId = orb.getPoa()->activate_object(servant.get());
  x3d::sai::Server_var ref = servant->_this();

  CosNaming::NamingContext_var context;
  CosNaming::Name label;
  label.length(1);

  try
  {
    // Fetch the root naming context
    CORBA::Object_var obj =
      orb.getCorbaOrb()->resolve_initial_references("NameService");
    context = CosNaming::NamingContext::_narrow(obj);
    if(CORBA::is_nil(context))
      ARCHON_THROW1(ServerNameException,
                    "CORBA naming service is not available.");
  }
  catch(CORBA::Exception &)
  {
    ARCHON_THROW1(ServerNameException,
                  "CORBA naming service is not available.");
```



```
  }

  try
  {
    label[0].id = "x3d";
    context = context->bind_new_context(label);
  }
  catch(CosNaming::NamingContext::AlreadyBound &ex)
  {
    CORBA::Object_var obj = context->resolve(label);
    context = CosNaming::NamingContext::_narrow(obj);
    if(CORBA::is_nil(context))
      ARCHON_THROW1(ServerNameException,
                    "'/x3d.context' is not a naming context.");
  }

  try
  {
    label[0].id = "sai";
    context = context->bind_new_context(label);
  }
  catch(CosNaming::NamingContext::AlreadyBound &ex)
  {
    CORBA::Object_var obj = context->resolve(label);
    context = CosNaming::NamingContext::_narrow(obj);
    if(CORBA::is_nil(context))
      ARCHON_THROW1(ServerNameException,
                    "'/x3d.context/sai.context' is not a naming "
                    "context.");
  }

  string name = server->getName();
  label[0].id = name.c_str();
  try
  {
    context->bind(label, ref);
  }
  catch(CosNaming::NamingContext::AlreadyBound &)
  {
    CORBA::Object_var boundObj = context->resolve(label);
    for(;;)
    {
      try
      {
        if(boundObj->_non_existent()) break;
      }
      catch(CORBA::Exception &) { break; }
      ARCHON_THROW1(ServerNameException,
                    "'/x3d.context/sai.context/" +
                    name + ".object' is in use.");
    }
    context->rebind(label, ref);
  }

  active = true;
}

void Listener::deactivate()
```

```
{
  if(!active) return;
  orb.getPoa()->deactivate_object(servantId);
  active = false;
}

void Listener::
startServingClientRequests(Ref<GroupingNode> applicationContainer)
{
  Mutex::Lock lock(mutex);
  this->applicationContainer = applicationContainer;
  activate();
}

CORBA::ULong Listener::getNodeTypeBitsInId()
{
  return NodeType::maxIndexBits;
}

CORBA::ULong Listener::getNodeType(const char *n)
{
  const NodeType *t = NodeType::lookup(n);
  if(!t) throw x3d::sai::InvalidNameException();
  return t->getIndex();
}

CORBA::ULong Listener::getImmediateField(CORBA::ULong ti, const char *n
    )
{
  if(ti >= NodeType::getTotalNumber())
    throw x3d::sai::InvalidTypeException();
  const NodeType *t = NodeType::get(ti);
  const FieldBase *f = t->lookupImmediateField(n);
  if(!f) throw x3d::sai::InvalidNameException();
  return f->getId();
}

x3d::sai::Session_ptr Listener::
createSession(x3d::sai::Application_ptr a)
{
  Ref<GroupingNode> g;
  {
    Mutex::Lock lock(mutex);
    g = applicationContainer;
  }
  Ref<Session> session = new Session(this, a, g);
  x3d::sai::Session_var ref =
    session->createServant(applicationHeartbeat);
  {
    Mutex::Lock lock(mutex);
    sessions.push_back(session);
  }
  return ref._retn();
}

void Listener::removeSession(Ref<Session> s)
{
  Mutex::Lock lock(mutex);
```



```
        sessions.remove(s);
      }

      void Listener::refForwardDestroy()
      {
        deactivate();
        sessions.clear();
        applicationContainer.reset();
      }
    }
  }
}
```

Listing: **archon/x3d/sai/orb.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */
#ifndef ARCHON_X3D_SAI_ORB_H
#define ARCHON_X3D_SAI_ORB_H

#include <omniORB4/CORBA.h>

#include <archon/util/thread.H>


namespace Archon
{
  namespace X3D
  {
    namespace SAI
    {
      using namespace Utilities;

      struct Orb
      {
        Orb(int port=-1);
        ~Orb();

        CORBA::ORB_var &getCorbaOrb()     { return orb; }
        PortableServer::POA_var &getPoa() { return poa; }

      private:
```

```
        CORBA::ORB_var orb;
        PortableServer::POA_var poa;
      };

      template<typename T>
      struct ServantRef
      {
        ServantRef(T *p): p(p) {}
        ~ServantRef() { p->_remove_ref(); }
        T *operator->() { return p; }
        T *get() const { return p; }
      private:
        T *p;
      };
    }
  }
}

#endif // ARCHON_X3D_SAI_ORB_H
```

Listing: **archon/x3d/sai/orb.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <iostream>

#include <archon/util/logger.H>
#include <archon/util/text.H>

#include <archon/x3d/sai/orb.H>

namespace Archon
{
  namespace X3D
  {
    namespace SAI
    {
      using namespace Utilities;
```



```
      Orb::Orb(int port)
      {
        char hostname[256];
        if(gethostname(hostname, sizeof(hostname))<0)
        {
          Logger::get()->log((string)strerror(errno) + "\n");
          abort();
        };

        hostname[sizeof(hostname)-1] = '\0';

        char endpoint[512];
        if(port<0) snprintf(endpoint, sizeof(endpoint), "giop:tcp:%s:",
             hostname);
        else snprintf(endpoint, sizeof(endpoint), "giop:tcp:%s:%d", hostname
             , port);

        const char *args[][2] =
          {
            { "endPoint"                , endpoint  },
          /*{ "offerBiDirectionalGIOP" , "1"       },
            { "acceptBiDirectionalGIOP", "1"       },
            { "clientTransportRule"    , "* tcp"   },
            { "serverTransportRule"    , "* tcp"   },*/
            { 0                        , 0         }
          };

        // Initialise the ORB.
        int argc = 0;
        orb = CORBA::ORB_init(argc, 0, "omniORB4", args);

        // Obtain a reference to the root POA.
        CORBA::Object_var obj = orb->resolve_initial_references("RootPOA");
        poa = PortableServer::POA::_narrow(obj);

        PortableServer::POA_var rootPoa = PortableServer::POA::_narrow(obj);

        CORBA::PolicyList policies(1);
        policies.length(1);
        policies[0] = new BiDirPolicy::BidirectionalPolicy(BiDirPolicy::BOTH
             );
        poa = rootPoa->create_POA("archon", PortableServer::POAManager::_nil
             (), policies);

        // Obtain a POAManager, and tell the POA to start accepting
        // requests on its objects.
        PortableServer::POAManager_var pman = poa->the_POAManager();
        pman->activate();
      }

      Orb::~Orb()
      {{ // The extra scope is needed to work around gcc3.2 bug #8287
        orb->destroy();
      }}
    }
  }
}
```

Listing: **archon/x3d/sai/route.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */
#ifndef ARCHON_X3D_SAI_ROUTE_H
#define ARCHON_X3D_SAI_ROUTE_H

#include <archon/x3d/server/event.H>

#include <archon/x3d/sai/session.H>

namespace Archon
{
  namespace X3D
  {
    namespace SAI
    {
      using namespace Utilities;

      /**
       * Base class for head of standard routes
       */
      struct ExternalRouteHead: virtual RefObjectBase
      {
        const BackRef<Session> session;
        const unsigned long cookie;

      protected:
        ExternalRouteHead(Ref<Session> session, unsigned long cookie):
          session(session), cookie(cookie) {}

        unsigned long getHeadCountNoLock() { return getUseCountNoLock(); }
      };

      struct ExternalRoute: RouteTail, ExternalRouteHead
      {
        struct Exception: virtual X3D::Exception {};
        struct ArgumentException: Exception, X3D::ArgumentException
        {
          ArgumentException() {}
          ArgumentException(string l, string m): Utilities::Exception(l, m
              ) {}
        };
```



```
        /**
         * Thrown by 'add' if the source field is not an event source or
         * if the target field is not an event target.
         */
        struct AccessException: ArgumentException
        {
          AccessException(string l): Utilities::Exception(l) {}
        };

        void cascadeEvent(const Event *e) const;

        static void add(Ref<Node> sourceNode, const FieldBase *sourceField,
                        Ref<Session> session, unsigned long cookie)
          throw(AccessException);

        static bool del(Ref<Node> sourceNode, const FieldBase *sourceField,
                        Ref<Session> session, unsigned long cookie);

        void refDispose(Mutex::Lock &l);

        ~ExternalRoute();

      private:
        ExternalRoute(Ref<Node> sourceNode, const FieldBase *sourceField,
                      Ref<Session> session, unsigned long cookie);

        bool match(const RouteTail *) const;
      };
    }
  }
}

#endif // ARCHON_X3D_SAI_ROUTE_H
```

Listing: **archon/x3d/sai/route.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#include <archon/x3d/server/scene.H>

#include <archon/x3d/sai/route.H>
```

```
namespace Archon
{
  namespace X3D
  {
    namespace SAI
    {
      using namespace Utilities;

      void ExternalRoute::cascadeEvent(const Event *e) const
      {
        try { session->handleExternalEvent(e->value.get(), cookie); }
        catch(ForwardDestroyedException &) {}
      };

      void ExternalRoute::add(Ref<Node> sourceNode, const FieldBase *
          sourceField,
                              Ref<Session> session, unsigned long cookie)
        throw(AccessException)
      {
        // Field input/output capabilities
        if(!sourceField->getIsEventSource()) ARCHON_THROW(AccessException);

        ExternalRoute *r =
          new ExternalRoute(sourceNode, sourceField, session, cookie);
        Ref<RouteTail> t(dynamic_cast<RouteTail *>(r));
        Ref<ExternalRouteHead> h(dynamic_cast<ExternalRouteHead *>(r));
        if(sourceField->addRoute(sourceNode.get(), t.get()))
          session->addRouteHead(h);
      }

      bool ExternalRoute::del(Ref<Node> sourceNode, const FieldBase *
          sourceField,
                              Ref<Session> session, unsigned long cookie)
      {
        ExternalRoute r(sourceNode, sourceField, session, cookie);
        return sourceField->delRouteMatch(sourceNode.get(), &r);
      }

      void ExternalRoute::refDispose(Mutex::Lock &l)
      {
        if(getTailCountNoLock() == 0)
        {
          if(getHeadCountNoLock() == 0)
          {
            l.release();
            delete this;
            return;
          }
          Ref<ExternalRouteHead> h(this, RefNoLockTag());
          l.release();
          try
          {
            session->removeRouteHead(h);
          }
          catch(ForwardDestroyedException &) {}
          return;
        }
```



```
        Ref<RouteTail> t(this, RefNoLockTag());
        l.release();
        try
        {
          Ref<Node> n = sourceNode.getRef();
          sourceField->delRoute(n.get(), t.get());
        }
        catch(ForwardDestroyedException &) {}
      }

      ExternalRoute::~ExternalRoute()
      {
      }

      ExternalRoute::ExternalRoute(Ref<Node> sourceNode, const FieldBase *
          sourceField,
                                   Ref<Session> session, unsigned long cookie
                                       ):
        RouteTail(sourceNode, sourceField),
        ExternalRouteHead(session, cookie)
      {
      }

      bool ExternalRoute::match(const RouteTail *r) const
      {
        const ExternalRoute *s = dynamic_cast<const ExternalRoute *>(r);
        return s && s->session == session &&
          s->cookie == cookie;
      }
    }
  }
}
```

Listing: **archon/x3d/sai/session.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */
#ifndef ARCHON_X3D_SAI_SESSION_H
#define ARCHON_X3D_SAI_SESSION_H

#include <vector>
#include <queue>
```

```
#include <archon/util/logger.H>
#include <archon/util/ref.H>
#include <archon/util/mutex.H>

#include <archon/x3d/server/load.H>

#include <archon/x3d/sai/orb.H>
#include <archon/x3d/sai/sai.H>
#include <archon/x3d/sai/value.H>
#include <archon/x3d/sai/change.H>

namespace Archon
{
  namespace X3D
  {
    namespace SAI
    {
      using namespace Utilities;

      struct Listener;
      struct ExternalRouteHead;

      struct Session: Porter, Loader::FileServer, Logger
      {
        Session(Ref<Listener> listener,
                x3d::sai::Application_ptr,
                Ref<GroupingNode> applicationContainer);

        ~Session();

        /**
         * Create the Session CORBA servant.
         *
         * \note Do not ignore the return value!!
         */
        x3d::sai::Session_ptr createServant(Time heartbeat);

        unsigned long node2id(Ref<Node>);
        Ref<Node> id2node(unsigned long) throw(IllegalNodeIdException);

        // Session (or Browser) services
        char *getServerName();
        CORBA::Double getCurrentFrameRate();
        CORBA::ULong createApplicationScene(const char *, CORBA::Boolean);
        CORBA::ULong loadApplicationScene(const char *, CORBA::Boolean);
        void launchApplicationScene(CORBA::ULong);
        void beginUpdate();
        void endUpdate();
        void print(const char *);
        void setSynchronizedChangesEnabled(CORBA::Boolean);
        void disposeBatch(const x3d::sai::DisposeSeq &,
                          const x3d::sai::DisposeSeq &);
        void disposeSession();

        // ExecutionContext services
        char *getBaseUri(CORBA::ULong);
        CORBA::ULong getRootGroup(CORBA::ULong);
```



```
        void setRootGroup(CORBA::ULong, CORBA::ULong);
        CORBA::ULong createNode(CORBA::ULong, const char *);
        CORBA::ULong createNode(CORBA::ULong, CORBA::ULong);
        CORBA::ULong getNode(CORBA::ULong, const char *);
        void addRoute(CORBA::ULong, CORBA::ULong, CORBA::ULong,
                      CORBA::ULong, CORBA::ULong);
        void delRoute(CORBA::ULong, CORBA::ULong, CORBA::ULong,
                      CORBA::ULong, CORBA::ULong);
        CORBA::ULong createGroupFromStream(CORBA::ULong,
                                           x3d::sai::StreamReader_ptr);
        void disposeContext(CORBA::ULong);

        // Node services
        char *getNodeType(CORBA::ULong);
        CORBA::ULong getField(CORBA::ULong, const char *);
        void disposeNode(CORBA::ULong);

        // Field services
        CORBA::ULong getFieldType(CORBA::ULong, CORBA::ULong);
        char *getFieldName(CORBA::ULong, CORBA::ULong);
        CORBA::ULong getFieldAccessType(CORBA::ULong, CORBA::ULong);
        void registerFieldInterest(CORBA::ULong, CORBA::ULong,
                                   CORBA::ULong);
        void unRegisterFieldInterest(CORBA::ULong, CORBA::ULong,
                                     CORBA::ULong);
        x3d::sai::Value *getFieldValue(CORBA::ULong, CORBA::ULong);
        void setFieldValue(CORBA::ULong, CORBA::ULong,
                           const x3d::sai::Value &);
        void disposeField(CORBA::ULong) {}

        // SequenceField services
        x3d::sai::Value *getFieldValueAt(CORBA::ULong, CORBA::ULong,
                                         CORBA::ULong);
        void setFieldValueAt(CORBA::ULong, CORBA::ULong, CORBA::ULong,
                             const x3d::sai::Value &);
        void setFieldValueAdd(CORBA::ULong, CORBA::ULong,
                              const x3d::sai::Value &);
        void setFieldValueDel(CORBA::ULong, CORBA::ULong,
                              const x3d::sai::Value &);

        void executeChanges();

        void handleChange(Ref<const ChangeBase>);

        Ref<GroupingNode> getApplicationContainer()
        {
          return applicationContainer;
        }

        void handleExternalEvent(const ValueBase *v, unsigned long cookie);

        void addRouteHead(Ref<const ExternalRouteHead>);
        void removeRouteHead(Ref<const ExternalRouteHead>);

        const BackRef<Listener> listener;

      private:
        void refForwardDestroy();
```

```
  Orb &orb;

  const x3d::sai::Application_var application;


  struct ExternalEvent;
  struct ExternalEventQueue;
  Ref<ExternalEventQueue> externalEventQueue;

  void pushEvents(const list<ExternalEvent> *);


  Mutex mutex;

  struct Servant;
  PortableServer::ObjectId_var servantId;
  bool active;

  /**
   * Activate the external event queue and the session CORBA
   * servant.
   *
   * NOTE: Do not ignore the return value!!
   */
  x3d::sai::Session_ptr activate(Time heartbeat);

  /**
   * Deactivate the Session CORBA servant.
   */
  void deactivate();

  RefMap<ExecutionContext> contextMap;
  RefMap<Node> nodeMap;

  queue<Ref<const ChangeBase> > changeQueue;

  /**
   * The group whos children are application subtrees.
   */
  Ref<GroupingNode> applicationContainer;

  bool buffered;
  bool synchronizedChanges;

  // Overriding from X3D::Loader::FileServer
  Ref<Stream::Reader> request(string)
    throw(FileServer::RequestException);

  // Overriding from Utilities::Logger
  /**
   * Send a log message to the aplication
   */
  void log(string) throw();

  Mutex routeHeadsMutex;
  list<Ref<const ExternalRouteHead> > routeHeads;
};
```



```
    }
  }
}

#endif // ARCHON_X3D_SAI_SESSION_H
```

Listing: **archon/x3d/sai/session.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#include <archon/util/cxx_demangle.H>
#include <archon/util/uri.H>

#include <archon/x3d/server/network.H>
#include <archon/x3d/server/parse_xml.H>

#include <archon/x3d/sai/exception.H>
#include <archon/x3d/sai/sai.H>
#include <archon/x3d/sai/session.H>
#include <archon/x3d/sai/listener.H>
#include <archon/x3d/sai/route.H>

#define CORBA_ENTRY_D(x) try { session->x; } \
catch(CORBA::UserException &) { throw; } \
catch(ForwardDestroyedException &) {} \
catch(...) { string m = SAI::Exception::catchShortExplanation(); \
throw x3d::sai::ServerErrorException(m.c_str()); }

#define CORBA_ENTRY_V(x) try { session->x; } \
catch(CORBA::UserException &) { throw; } \
catch(ForwardDestroyedException &) \
{ throw x3d::sai::InvalidSessionException(); } \
catch(...) { string m = SAI::Exception::catchShortExplanation(); \
throw x3d::sai::ServerErrorException(m.c_str()); }

#define CORBA_ENTRY_R(x) try { return session->x; } \
catch(CORBA::UserException &) { throw; } \
catch(ForwardDestroyedException &) \
{ throw x3d::sai::InvalidSessionException(); } \
catch(...) { string m = SAI::Exception::catchShortExplanation(); \
throw x3d::sai::ServerErrorException(m.c_str()); }
```

```
namespace Archon
{
  namespace X3D
  {
    namespace SAI
    {
      using namespace Utilities;

      struct Session::Servant:
        POA_x3d::sai::Session, PortableServer::RefCountServantBase
      {
        const BackRef<SAI::Session> session;

        Servant(Ref<SAI::Session> session): session(session) {}

        ~Servant()
        {
        }

        // Session (or Browser) services

        char *getServerName()
        {
          CORBA_ENTRY_R(getServerName());
        }

        CORBA::Double getCurrentFrameRate()
        {
          CORBA_ENTRY_R(getCurrentFrameRate());
        }

        CORBA::ULong createApplicationScene(const char *baseUri,
                                            CORBA::Boolean localFileUri)
        {
          CORBA_ENTRY_R(createApplicationScene(baseUri, localFileUri));
        }

        CORBA::ULong loadApplicationScene(const char *uri,
                                          CORBA::Boolean localFileUri)
        {
          CORBA_ENTRY_R(loadApplicationScene(uri, localFileUri));
        }

        void launchApplicationScene(CORBA::ULong contextId)
        {
          CORBA_ENTRY_V(launchApplicationScene(contextId));
        }

        void beginUpdate()
        {
          CORBA_ENTRY_V(beginUpdate());
        }

        void endUpdate()
        {
          CORBA_ENTRY_V(endUpdate());
        }
```



```
        void print(const char *s)
        {
          CORBA_ENTRY_V(print(s));
        }

        void setSynchronizedChangesEnabled(CORBA::Boolean enabled)
        {
          CORBA_ENTRY_V(setSynchronizedChangesEnabled(enabled));
        }

        void disposeBatch(const x3d::sai::DisposeSeq &contexts,
                          const x3d::sai::DisposeSeq &nodes)
        {
          CORBA_ENTRY_V(disposeBatch(contexts, nodes));
        }

        void disposeSession()
        {
          CORBA_ENTRY_D(disposeSession());
        }


        // ExecutionContext services

        char *getBaseUri(CORBA::ULong contextId)
        {
          CORBA_ENTRY_R(getBaseUri(contextId));
        }

        CORBA::ULong getRootGroup(CORBA::ULong contextId)
        {
          CORBA_ENTRY_R(getRootGroup(contextId));
        }

        void setRootGroup(CORBA::ULong contextId, CORBA::ULong nodeId)
        {
          CORBA_ENTRY_V(setRootGroup(contextId, nodeId));
        }

        CORBA::ULong createNode(CORBA::ULong contextId, const char *typeName)
        {
          CORBA_ENTRY_R(createNode(contextId, typeName));
        }

        CORBA::ULong createNodeFast(CORBA::ULong contextId, CORBA::ULong
             typeId)
        {
          CORBA_ENTRY_R(createNode(contextId, typeId));
        }

        CORBA::ULong getNode(CORBA::ULong contextId, const char *nodeName)
        {
          CORBA_ENTRY_R(getNode(contextId, nodeName));
        }

        void addRoute(CORBA::ULong contextId,
                      CORBA::ULong fromNodeId, CORBA::ULong fromFieldId,
```

```
              CORBA::ULong toNodeId, CORBA::ULong toFieldId)
{
  CORBA_ENTRY_V(addRoute(contextId, fromNodeId, fromFieldId,
                   toNodeId, toFieldId));
}

void delRoute(CORBA::ULong contextId,
              CORBA::ULong fromNodeId, CORBA::ULong fromFieldId,
              CORBA::ULong toNodeId, CORBA::ULong toFieldId)
{
  CORBA_ENTRY_V(delRoute(contextId, fromNodeId, fromFieldId,
                   toNodeId, toFieldId));
}

CORBA::ULong createGroupFromStream(CORBA::ULong contextId,
                                   x3d::sai::StreamReader_ptr r)
{
  CORBA_ENTRY_R(createGroupFromStream(contextId, r));
}

void disposeContext(CORBA::ULong contextId)
{
  CORBA_ENTRY_V(disposeContext(contextId));
}


// Node services

char *getNodeType(CORBA::ULong nodeId)
{
  CORBA_ENTRY_R(getNodeType(nodeId));
}

CORBA::ULong getField(CORBA::ULong nodeId, const char *name)
{
  CORBA_ENTRY_R(getField(nodeId, name));
}

void disposeNode(CORBA::ULong nodeId)
{
  CORBA_ENTRY_V(disposeNode(nodeId));
}


// Field services

CORBA::ULong getFieldType(CORBA::ULong nodeId, CORBA::ULong fieldId)
{
  CORBA_ENTRY_R(getFieldType(nodeId, fieldId));
}

char *getFieldName(CORBA::ULong nodeId, CORBA::ULong fieldId)
{
  CORBA_ENTRY_R(getFieldName(nodeId, fieldId));
}

CORBA::ULong getFieldAccessType(CORBA::ULong nodeId,
                                CORBA::ULong fieldId)
```



```
{
  CORBA_ENTRY_R(getFieldAccessType(nodeId, fieldId));
}

void registerFieldInterest(CORBA::ULong nodeId, CORBA::ULong fieldId,
                           CORBA::ULong cookie)
{
  CORBA_ENTRY_V(registerFieldInterest(nodeId, fieldId, cookie));
}

void unRegisterFieldInterest(CORBA::ULong nodeId,
                             CORBA::ULong fieldId,
                             CORBA::ULong cookie)
{
  CORBA_ENTRY_V(unRegisterFieldInterest(nodeId, fieldId, cookie));
}

x3d::sai::Value *getFieldValue(CORBA::ULong nodeId,
                               CORBA::ULong fieldId)
{
  CORBA_ENTRY_R(getFieldValue(nodeId, fieldId));
}

void setFieldValue(CORBA::ULong nodeId, CORBA::ULong fieldId,
                   const x3d::sai::Value &value)
{
  CORBA_ENTRY_V(setFieldValue(nodeId, fieldId, value));
}

void disposeField(CORBA::ULong fieldId)
{
  CORBA_ENTRY_V(disposeField(fieldId));
}


// Field services

x3d::sai::Value *getFieldValueAt(CORBA::ULong nodeId,
                                 CORBA::ULong fieldId,
                                 CORBA::ULong index)
{
  CORBA_ENTRY_R(getFieldValueAt(nodeId, fieldId, index));
}

void setFieldValueAt(CORBA::ULong nodeId, CORBA::ULong fieldId,
                     CORBA::ULong index, const x3d::sai::Value &value
                          )
{
  CORBA_ENTRY_V(setFieldValueAt(nodeId, fieldId, index, value));
}

void setFieldValueAdd(CORBA::ULong nodeId, CORBA::ULong fieldId,
                      const x3d::sai::Value &value)
{
  CORBA_ENTRY_V(setFieldValueAdd(nodeId, fieldId, value));
}

void setFieldValueDel(CORBA::ULong nodeId, CORBA::ULong fieldId,
```

```
                          const x3d::sai::Value &value)
  {
    CORBA_ENTRY_V(setFieldValueDel(nodeId, fieldId, value));
  }
};


struct Reader: Stream::ReaderBase<char>
{
  x3d::sai::StreamReader_var reader;

  bool eoi;

  Reader(x3d::sai::StreamReader_ptr r): eoi(false)
  {
    reader = x3d::sai::StreamReader::_duplicate(r);
  }

  ~Reader()
  {{
    try { reader->dispose(); }
    catch(...) {}
  }}

  int read(char *buffer, int size)
    throw(Stream::ReadException, ThreadTerminatedException)
  {
    if(eoi) return 0;
    x3d::sai::chararray_var a;
    try { a = reader->read(size); }
    catch(x3d::sai::StreamReadException &e)
    {
      ARCHON_THROW1(Stream::ReadException, string(e.message));
    }
    catch(...)
    {
      ARCHON_THROW1(Stream::ReadException, "Caught unknown exception");
    }
    if(a->length()>(unsigned)size)
      ARCHON_THROW1(Stream::ReadException, "Too much data");
    if(a->length()<(unsigned)size) eoi = true;
    for(unsigned i=0; i<a->length(); ++i) buffer[i] = a[i];
    return a->length();
  }
};


struct ApplicationContext: ExecutionContext
{
  ApplicationContext(Ref<Server> server, Uri baseUri,
                     Ref<Session> fileServer):
    ExecutionContext(server, baseUri, fileServer) {}

  ~ApplicationContext()
  {
  }

  BackRef<Transform> transform;
```



```
  Ref<Transform> getTransform()
  {
    Mutex::Lock l(contextMutex);
    try { return transform.getRef(); }
    catch(ForwardDestroyedException &) { return 0; }
  }

  bool setTransform(Ref<Transform> t)
  {
    Mutex::Lock l(contextMutex);
    if(transform) return false;
    transform = t;
    return true;
  }
};

struct ApplicationContextReaper: Thread
{
  ApplicationContextReaper(Ref<GroupingNode> applicationContainer):
    applicationContainer(applicationContainer) {}

  void main()
  {
    Time time;
    try
    {
      time = applicationContainer->context->server->getNextTimeStamp();
    }
    catch(ForwardDestroyedException &) { return; }
    vector<Ref<ExecutionContext> >::iterator i = contexts.begin();
    while(i != contexts.end())
    {
      ExecutionContext *c = i->get();
      ++i;
      ApplicationContext *a = dynamic_cast<ApplicationContext *>(c);
      if(!a) continue;
      Ref<Transform> t = a->getTransform();
      if(!t) continue;
      applicationContainer->remove(GroupingNode::childrenField,
                                   t, time);
    }
  }

  Ref<GroupingNode> applicationContainer;
  vector<Ref<ExecutionContext> > contexts;
};

struct Session::ExternalEvent
{
  const Ref<const ValueBase> value;
  const unsigned long cookie;

  ExternalEvent(const ValueBase *value, unsigned long cookie):
    value(value), cookie(cookie) {}
};

struct Session::ExternalEventQueue: Thread
```

```
{
  ExternalEventQueue(Ref<Session>, Time heartbeat);

  ~ExternalEventQueue()
  {
  }

  const BackRef<Session> session;
  Time heartbeat;

  Mutex mutex;
  Condition nonEmpty;
  list<ExternalEvent> eventQueue;
  bool hasTerminated;
  bool flushRequest;
  bool flushing;
  Condition flushed;

  void main();

  void addEvent(const ValueBase *, unsigned long cookie);

  /**
   * Wait until the contents of the queue at the time of
   * calling is received by the application.
   */
  void flush();
};


Session::Session(Ref<Listener> listener,
                 x3d::sai::Application_ptr a,
                 Ref<GroupingNode> applicationContainer):
  listener(listener),
  orb(listener->getOrb()),
  application(x3d::sai::Application::_duplicate(a)),
  active(false),
  applicationContainer(applicationContainer),
  buffered(false),
  synchronizedChanges(true)
{
}

Session::~Session()
{{
}}


void Session::refForwardDestroy()
{
  /*
  try { listener->server->getLogger()->log(nodeMap.disp()); }
  catch(...) {}
  */
  externalEventQueue->terminate(); // Stop async activity
  externalEventQueue.reset();
  deactivate();
  nodeMap.clear();
```



```
  Ref<ApplicationContextReaper> r =
    new ApplicationContextReaper(applicationContainer);
  contextMap.clear(&r->contexts);
  Thread::start(r);
  applicationContainer.reset();
  routeHeads.clear();
}

x3d::sai::Session_ptr Session::createServant(Time heartbeat)
{
  Mutex::Lock lock(mutex);
  return activate(heartbeat);
}

x3d::sai::Session_ptr Session::activate(Time heartbeat)
{
  if(active) deactivate();
  externalEventQueue = new ExternalEventQueue(this, heartbeat);
  Thread::start(externalEventQueue);
  ServantRef<Servant> servant = new Servant(this);
  servantId = orb.getPoa()->activate_object(servant.get());
  x3d::sai::Session_var ref = servant->_this();
  active = true;
  return ref._retn();
}

void Session::deactivate()
{
  if(!active) return;
  orb.getPoa()->deactivate_object(servantId);
  active = false;
}

void Session::executeChanges()
{
  Time time = listener->server->getNextTimeStamp();
  while(!changeQueue.empty())
  {
    try
    {
      changeQueue.front()->apply(time);
    }
    catch(Node::AccessException &) {}
    catch(Node::RangeException &) {}
    changeQueue.pop();
  }
}


unsigned long Session::node2id(Ref<Node> n)
{
  if(!n) return 0;
  long t = n->getType()->getIndex();
  if(n>255) ARCHON_THROW1(InternalException, "Node index overflow");
  return static_cast<unsigned long>(nodeMap.get(n)) <<
    NodeType::maxIndexBits | t;
}
```

```
Ref<Node> Session::id2node(unsigned long i)
  throw(IllegalNodeIdException)
{
  if(!i) return 0;
  Ref<Node> n = nodeMap(i >> NodeType::maxIndexBits);
  if(!n) ARCHON_THROW(Porter::IllegalNodeIdException);
  return n;
}


// Session (or Browser) services

char *Session::getServerName()
{
  string n = listener->server->getName();
  return CORBA::string_dup(n.c_str());
}

CORBA::Double Session::getCurrentFrameRate()
{
  return listener->server->getFrameRate();
}

CORBA::ULong
Session::createApplicationScene(const char *u,
                                CORBA::Boolean localFileUri)
{
  Uri baseUri(u);

  Ref<ApplicationContext> context =
    new ApplicationContext(listener->server.getRef(),
                           baseUri, localFileUri ? this : 0);

  return contextMap.get(context, true);
}

CORBA::ULong
Session::loadApplicationScene(const char *u,
                              CORBA::Boolean localFileUri)
{
  Uri uri(u);

  Ref<ApplicationContext> context =
    new ApplicationContext(listener->server.getRef(),
                           uri, localFileUri ? this : 0);

  Ref<Loader> loader = listener->server->getLoader();
  if(loader) try
  {
    Ref<Web::Client::Response> response = loader->getWebClient()->
      request(Web::Client::Request::get(uri));
    Ref<Group> rootGroup = new Group(Ref<ExecutionContext>(context));
    XML::parse(context, rootGroup, response,
               false, this, uri.getFile());
    context->setRootGroup(rootGroup);
  }
  catch(Web::RequestException &e)
  {
```



```
    string message = e.getMessage();
    throw x3d::sai::FileRequestException(message.c_str());
  }
  catch(Parser::ParseException &)
  {
    throw x3d::sai::InvalidX3DException();
  }

  return contextMap.get(context, true);
}

void Session::launchApplicationScene(CORBA::ULong ci)
{
  // Subscribe
  Ref<ExecutionContext> c = contextMap(ci);
  if(!c) throw x3d::sai::InvalidExecutionContextException();
  Ref<ApplicationContext> applicationContext =
    dynamic_cast<ApplicationContext *>(c.get());
  if(!applicationContext)
    throw x3d::sai::InvalidExecutionContextException();
  Ref<ExecutionContext> applicationContainerContext =
    applicationContainer->context.getRef();

  // Create glue
  Ref<ApplicationLink> applicationLink =
    new ApplicationLink(applicationContainerContext);
  Ref<Transform> transform =
    new Transform(applicationContainerContext);

  if(!applicationContext->setTransform(transform))
    throw x3d::sai::AlreadyLaunchedException();

  // Combine
  Time t = listener->server->getNextTimeStamp();
  applicationLink->setSubContext(c);
  transform->add(GroupingNode::childrenField, applicationLink, t);
  applicationContainer->add(GroupingNode::childrenField, transform, t);
}

void Session::beginUpdate()
{
  Mutex::Lock lock(mutex);
  buffered = true;
}

void Session::endUpdate()
{
  Mutex::Lock lock(mutex);
  executeChanges();
  buffered = false;
}

void Session::print(const char *s)
{
  listener->server->print(s);
}

void Session::setSynchronizedChangesEnabled(CORBA::Boolean enabled)
```

```
{
  Mutex::Lock lock(mutex);
  synchronizedChanges = enabled;
}

void Session::disposeBatch(const x3d::sai::DisposeSeq &c,
                           const x3d::sai::DisposeSeq &n)
{
  for(unsigned i=0; i<c.length(); ++i)
  {
    const x3d::sai::DisposeEntry &e = c[i];
    contextMap.remove(e.id, e.transCount);
  }

  for(unsigned i=0; i<n.length(); ++i)
  {
    const x3d::sai::DisposeEntry &e = n[i];
    nodeMap.remove(e.id >> NodeType::maxIndexBits, e.transCount);
  }
}

/*
 * \todo WE MUST MAKE SURE THAT THE APPLICATION IS NOT CONTACTED
 * AT ALL AFTER THE DISPOSE_SESSION REQUEST HAS RETURNED TO THE
 * APPLICATION. THIS INCLUDES EVENT HANDLING, URI REQUESTING AND
 * LOGGING.
 */
void Session::disposeSession()
{
  deactivate();
  try { listener->removeSession(this); }
  catch(ForwardDestroyedException &) {}
}


// ExecutionContext services

char *Session::getBaseUri(CORBA::ULong ci)
{
  Ref<ExecutionContext> c = contextMap(ci);
  if(!c) throw x3d::sai::InvalidExecutionContextException();
  return CORBA::string_dup(c->getBaseUri().toString().c_str());
}

CORBA::ULong Session::getRootGroup(CORBA::ULong ci)
{
  Ref<ExecutionContext> c = contextMap(ci);
  if(!c) throw x3d::sai::InvalidExecutionContextException();
  return node2id(c->getRootGroup());
}

void Session::setRootGroup(CORBA::ULong ci, CORBA::ULong ni)
{
  Ref<ExecutionContext> c = contextMap(ci);
  if(!c) throw x3d::sai::InvalidExecutionContextException();
  try
  {
    Ref<Node> n = id2node(ni);
```



```
    Ref<Group> g = dynamic_cast<Group *>(n.get());
    if(!g && n) throw x3d::sai::InvalidNodeException();
    c->setRootGroup(g);
  }
  catch(IllegalNodeIdException &)
  {
    throw x3d::sai::InvalidNodeException();
  }
}

CORBA::ULong Session::createNode(CORBA::ULong ci, const char *n)
{
  Ref<ExecutionContext> c = contextMap(ci);
  if(!c) throw x3d::sai::InvalidExecutionContextException();
  const NodeType *t = NodeType::lookup(n);
  if(!t || !t->isUserInstantiable())
    throw x3d::sai::InvalidNameException();
  return node2id(t->instantiate(c));
}

CORBA::ULong Session::createNode(CORBA::ULong ci, CORBA::ULong ti)
{
  Ref<ExecutionContext> c = contextMap(ci);
  if(!c) throw x3d::sai::InvalidExecutionContextException();
  if(ti >= NodeType::getTotalNumber())
    throw x3d::sai::InvalidTypeException();
  const NodeType *t = NodeType::get(ti);
  if(!t->isUserInstantiable())
    throw x3d::sai::InvalidTypeException();
  return node2id(t->instantiate(c));
}

CORBA::ULong Session::getNode(CORBA::ULong ci, const char *n)
{
  Ref<ExecutionContext> c = contextMap(ci);
  if(!c) throw x3d::sai::InvalidExecutionContextException();
  Ref<Node> node = c->lookupNode(n);
  if(!node) throw x3d::sai::InvalidNameException();
  return node2id(node);
}

void Session::addRoute(CORBA::ULong ci,
                       CORBA::ULong fni, CORBA::ULong ffi,
                       CORBA::ULong tni, CORBA::ULong tfi)
{
  Ref<ExecutionContext> c = contextMap(ci);
  if(!c) throw x3d::sai::InvalidExecutionContextException();
  try
  {
    Ref<Node> sourceNode = id2node(fni);
    const FieldBase *sourceField =
      FieldBase::fetch(ffi, sourceNode.get());
    Ref<Node> targetNode = id2node(tni);
    const FieldBase *targetField =
      FieldBase::fetch(tfi, targetNode.get());
    if(!sourceField || !targetField)
      throw x3d::sai::InvalidFieldException();
    BackRef<ExecutionContext> d = c;
```

```
    if(sourceNode->context != d || targetNode->context != d)
      throw x3d::sai::InvalidNodeException();

    // Field input/output capabilities
    if(!sourceField->getIsEventSource() ||
       !targetField->getIsEventTarget())
      throw x3d::sai::InvalidAccessTypeException();

    // Type conformance
    if(sourceField->getType() != targetField->getType())
      throw x3d::sai::InvalidAccessTypeException();

    // Injective node type restriction
    if(const NodeFieldBase *s =
       dynamic_cast<const NodeFieldBase *>(sourceField))
    {
      const NodeFieldBase *t =
        dynamic_cast<const NodeFieldBase *>(targetField);
      if(!s->getNodeType()->isDerivedFrom(t->getNodeType()))
        throw x3d::sai::InvalidAccessTypeException();
    }

    handleChange(new RouteChange(sourceNode, sourceField,
                                 targetNode, targetField));
  }
  catch(IllegalNodeIdException &)
  {
    throw x3d::sai::InvalidNodeException();
  }
}

void Session::delRoute(CORBA::ULong ci,
                       CORBA::ULong fni, CORBA::ULong ffi,
                       CORBA::ULong tni, CORBA::ULong tfi)
{
  Ref<ExecutionContext> c = contextMap(ci);
  if(!c) throw x3d::sai::InvalidExecutionContextException();
  try
  {
    Ref<Node> sourceNode = id2node(fni);
    const FieldBase *sourceField =
      FieldBase::fetch(ffi, sourceNode.get());
    Ref<Node> targetNode = id2node(tni);
    const FieldBase *targetField =
      FieldBase::fetch(tfi, targetNode.get());
    if(!sourceField || !targetField)
      throw x3d::sai::InvalidFieldException();
    BackRef<ExecutionContext> d = c;
    if(sourceNode->context != d || targetNode->context != d)
      throw x3d::sai::InvalidNodeException();

    // Field input/output capabilities
    if(!sourceField->getIsEventSource() ||
       !targetField->getIsEventTarget())
      throw x3d::sai::InvalidAccessTypeException();

    // Type conformance
    if(sourceField->getType() != targetField->getType())
```



```
      throw x3d::sai::InvalidAccessTypeException();

    // Injective node type restriction
    if(const NodeFieldBase *s =
       dynamic_cast<const NodeFieldBase *>(sourceField))
    {
      const NodeFieldBase *t =
        dynamic_cast<const NodeFieldBase *>(targetField);
      if(!s->getNodeType()->isDerivedFrom(t->getNodeType()))
        throw x3d::sai::InvalidAccessTypeException();
    }

    handleChange(new RouteChange(sourceNode, sourceField,
                                 targetNode, targetField, true));
  }
  catch(IllegalNodeIdException &)
  {
    throw x3d::sai::InvalidNodeException();
  }
}

CORBA::ULong Session::createGroupFromStream(CORBA::ULong ci,
                                            x3d::sai::StreamReader_ptr
                                                r)
{
  Ref<ExecutionContext> c = contextMap(ci);
  if(!c) throw x3d::sai::InvalidExecutionContextException();
  Ref<GroupingNode> g = new Group(c);
  try
  {
    XML::parse(c, g, new Reader(r), false, this, "<SAI parse request>"
         );
  }
  catch(Parser::ParseException &)
  {
    throw x3d::sai::InvalidX3DException();
  }
  return node2id(g);
}

void Session::disposeContext(CORBA::ULong ci)
{
  contextMap.remove(ci);
}


// Node services

char *Session::getNodeType(CORBA::ULong ni)
{
  if(!ni) throw x3d::sai::InvalidNodeException();
  try
  {
    return CORBA::string_dup(id2node(ni)->getType()->getName().c_str
         ());
  }
  catch(IllegalNodeIdException &)
  {
```

```
    throw x3d::sai::InvalidNodeException();
  }
}

CORBA::ULong Session::getField(CORBA::ULong ni, const char *n)
{
  if(!ni) throw x3d::sai::InvalidNodeException();
  try
  {
    const FieldBase *f = id2node(ni)->lookupField(n);
    if(!f) throw x3d::sai::InvalidNameException();
    return f->getId();
  }
  catch(IllegalNodeIdException &)
  {
    throw x3d::sai::InvalidNodeException();
  }
}

void Session::disposeNode(CORBA::ULong ni)
{
  nodeMap.remove(ni);
}


// Field services

CORBA::ULong Session::getFieldType(CORBA::ULong ni, CORBA::ULong fi)
{
  try
  {
    Ref<Node> n = id2node(ni);
    if(!n) throw x3d::sai::InvalidNodeException();
    const FieldBase *f = FieldBase::fetch(fi, n.get());
    if(!f) throw x3d::sai::InvalidFieldException();
    return f->getType()->getIndex();
  }
  catch(IllegalNodeIdException &)
  {
    throw x3d::sai::InvalidNodeException();
  }
}

char *Session::getFieldName(CORBA::ULong ni, CORBA::ULong fi)
{
  try
  {
    Ref<Node> n = id2node(ni);
    if(!n) throw x3d::sai::InvalidNodeException();
    const FieldBase *f = FieldBase::fetch(fi, n.get());
    if(!f) throw x3d::sai::InvalidFieldException();
    return CORBA::string_dup(f->getName().c_str());
  }
  catch(IllegalNodeIdException &)
  {
    throw x3d::sai::InvalidNodeException();
  }
}
```



```
CORBA::ULong Session::getFieldAccessType(CORBA::ULong ni, CORBA::ULong
     fi)
{
  try
  {
    Ref<Node> n = id2node(ni);
    if(!n) throw x3d::sai::InvalidNodeException();
    const FieldBase *f = FieldBase::fetch(fi, n.get());
    if(!f) throw x3d::sai::InvalidFieldException();
    return
      f->getIsEventSource() ?
      f->getIsEventTarget() ? 3 : 2 :
      f->getIsEventTarget() ? 1 : 0;
  }
  catch(IllegalNodeIdException &)
  {
    throw x3d::sai::InvalidNodeException();
  }
}

void Session::registerFieldInterest(CORBA::ULong ni, CORBA::ULong fi,
                                    CORBA::ULong c)
{
  try
  {
    Ref<Node> n = id2node(ni);
    if(!n) throw x3d::sai::InvalidNodeException();
    const FieldBase *f = FieldBase::fetch(fi, n.get());
    if(!f) throw x3d::sai::InvalidFieldException();

    // Field input/output capabilities
    if(!f->getIsEventSource())
      throw x3d::sai::InvalidAccessTypeException();

    ExternalRoute::add(n, f, this, c);
  }
  catch(IllegalNodeIdException &)
  {
    throw x3d::sai::InvalidNodeException();
  }
}

/**
 * MUST FLUSH OUTBOUND EVENT QUEUE WHERE AN
 * UNREGISTER_FIELD_INTEREST REQUEST IS MADE. THIS IS IMPORTANT
 * FOR THE APPLICATION, SINCE IT NEEDS TO KNOW THAT THE REQUEST
 * IS COMPLETE THEN NO MORE EVENTS FROM THAT SOURCE CAN ARRIVE.
 */
void Session::unRegisterFieldInterest(CORBA::ULong ni, CORBA::ULong fi,
                                      CORBA::ULong c)
{
  try
  {
    Ref<Node> n = id2node(ni);
    if(!n) throw x3d::sai::InvalidNodeException();
    const FieldBase *f = FieldBase::fetch(fi, n.get());
    if(!f) throw x3d::sai::InvalidFieldException();
```

```
    // Field input/output capabilities
    if(!f->getIsEventSource())
      throw x3d::sai::InvalidAccessTypeException();

    ExternalRoute::del(n, f, this, c);
    externalEventQueue->flush();
  }
  catch(IllegalNodeIdException &)
  {
    throw x3d::sai::InvalidNodeException();
  }
}

x3d::sai::Value *Session::getFieldValue(CORBA::ULong ni, CORBA::ULong
     fi)
{
  try
  {
    Ref<Node> n = id2node(ni);
    if(!n) throw x3d::sai::InvalidNodeException();
    const FieldBase *f = FieldBase::fetch(fi, n.get());
    if(!f) throw x3d::sai::InvalidFieldException();
    if(!f->getIsEventSource() && f->getIsEventTarget())
      throw x3d::sai::InvalidAccessTypeException();
    Ref<ValueBase> v = n->get(f);
    x3d::sai::Value_var w = new x3d::sai::Value();
    ex(v.get(), w.inout());
    return w._retn();
  }
  catch(IllegalNodeIdException &)
  {
    throw x3d::sai::InvalidNodeException();
  }
  catch(Node::AccessException &)
  {
    throw x3d::sai::InvalidOperationTimingException();
  }
}

/**
 * \todo Check SFNode and MFNode values for context
 * relationship, and possibly issue a NodeInUseException.
 *
 * \todo Check SFNode and MFNode values for containing null
 * references, and issue an IllegalNodeException.
 */
void Session::setFieldValue(CORBA::ULong ni, CORBA::ULong fi,
                            const x3d::sai::Value &v)
{
  try
  {
    Ref<Node> n = id2node(ni);
    if(!n) throw x3d::sai::InvalidNodeException();
    const FieldBase *f = FieldBase::fetch(fi, n.get());
    if(!f) throw x3d::sai::InvalidFieldException();
    if(f->getIsEventSource() && !f->getIsEventTarget())
      throw x3d::sai::InvalidAccessTypeException();
```



```
    Ref<ValueBase> w = im(v);
    if(w->getFieldType() != f->getType())
      throw x3d::sai::InvalidValueException();
    handleChange(new FieldSetChange(n, f, w));
  }
  catch(TypeMismatchException &)
  {
    throw x3d::sai::InvalidValueException();
  }
  catch(IllegalNodeIdException &)
  {
    throw x3d::sai::InvalidNodeException();
  }
  catch(Node::AccessException &)
  {
    throw x3d::sai::InvalidOperationTimingException();
  }
}


// SequenceField services

x3d::sai::Value *Session::getFieldValueAt(CORBA::ULong ni,
                                          CORBA::ULong fi,
                                          CORBA::ULong i)
{
  try
  {
    Ref<Node> n = id2node(ni);
    if(!n) throw x3d::sai::InvalidNodeException();
    const FieldBase *f = FieldBase::fetch(fi, n.get());
    if(!f) throw x3d::sai::InvalidFieldException();
    if(!f->getIsEventSource() && f->getIsEventTarget())
      throw x3d::sai::InvalidAccessTypeException();
    const SequenceFieldBase *s =
      dynamic_cast<const SequenceFieldBase *>(f);
    if(!s) throw x3d::sai::InvalidAccessTypeException();
    Ref<ValueBase> v = n->getAt(s, i);
    x3d::sai::Value_var w = new x3d::sai::Value();
    ex(v.get(), w.inout());
    return w._retn();
  }
  catch(IllegalNodeIdException &)
  {
    throw x3d::sai::InvalidNodeException();
  }
  catch(Node::AccessException &)
  {
    throw x3d::sai::InvalidOperationTimingException();
  }
  catch(Node::RangeException &)
  {
    throw x3d::sai::InvalidIndexException();
  }
}

/**
 * \todo Check SFNode values for context relationship, and
```

```
 * possibly issue a NodeInUseException.
 *
 * \todo Check SFNode values for being null, and issue an
 * IllegalNodeException.
 */
void Session::setFieldValueAt(CORBA::ULong ni, CORBA::ULong fi,
                              CORBA::ULong i, const x3d::sai::Value &v)
{
  try
  {
    Ref<Node> n = id2node(ni);
    if(!n) throw x3d::sai::InvalidNodeException();
    const FieldBase *f = FieldBase::fetch(fi, n.get());
    if(!f) throw x3d::sai::InvalidFieldException();
    if(f->getIsEventSource() != f->getIsEventTarget())
      throw x3d::sai::InvalidAccessTypeException();
    const SequenceFieldBase *s =
      dynamic_cast<const SequenceFieldBase *>(f);
    if(!s) throw x3d::sai::InvalidAccessTypeException();
    Ref<ValueBase> w = im(v);
    if(w->getFieldType() != f->getType()->getSingleType())
      throw x3d::sai::InvalidValueException();
    handleChange(new FieldSetAtChange(n, s, w, i));
  }
  catch(TypeMismatchException &)
  {
    throw x3d::sai::InvalidValueException();
  }
  catch(IllegalNodeIdException &)
  {
    throw x3d::sai::InvalidNodeException();
  }
  catch(Node::AccessException &)
  {
    // Note: This exception will not be utilized until we
    // implement a synchronized setting mechanism
    throw x3d::sai::InvalidOperationTimingException();
  }
  catch(Node::RangeException &)
  {
    // Note: This exception will not be utilized until we
    // implement a synchronized setting mechanism
    throw x3d::sai::InvalidIndexException();
  }
}

/**
 * \todo Check SFNode values for context relationship, and
 * possibly issue a NodeInUseException.
 *
 * \todo Check SFNode values for being null, and issue an
 * IllegalNodeException.
 */
void Session::setFieldValueAdd(CORBA::ULong ni, CORBA::ULong fi,
                               const x3d::sai::Value &v)
{
  try
  {
```



```
    Ref<Node> n = id2node(ni);
    if(!n) throw x3d::sai::InvalidNodeException();
    const FieldBase *f = FieldBase::fetch(fi, n.get());
    if(!f) throw x3d::sai::InvalidFieldException();
    if(f->getIsEventSource() != f->getIsEventTarget())
      throw x3d::sai::InvalidAccessTypeException();
    const SequenceFieldBase *s =
      dynamic_cast<const SequenceFieldBase *>(f);
    if(!s) throw x3d::sai::InvalidAccessTypeException();
    Ref<ValueBase> w = im(v);
    if(w->getFieldType() != f->getType()->getSingleType())
      throw x3d::sai::InvalidValueException();
    handleChange(new FieldListChange(n, s, w));
  }
  catch(TypeMismatchException &)
  {
    throw x3d::sai::InvalidValueException();
  }
  catch(IllegalNodeIdException &)
  {
    throw x3d::sai::InvalidNodeException();
  }
  catch(Node::AccessException &)
  {
    // Note: This exception will not be utilized until we
    // implement a synchronized setting mechanism
    throw x3d::sai::InvalidOperationTimingException();
  }
}

/**
 * \todo Check SFNode values for context relationship, and
 * possibly issue a NodeInUseException.
 *
 * \todo Check SFNode values for being null, and issue an
 * IllegalNodeException.
 */
void Session::setFieldValueDel(CORBA::ULong ni, CORBA::ULong fi,
                               const x3d::sai::Value &v)
{
  try
  {
    Ref<Node> n = id2node(ni);
    if(!n) throw x3d::sai::InvalidNodeException();
    const FieldBase *f = FieldBase::fetch(fi, n.get());
    if(!f) throw x3d::sai::InvalidFieldException();
    if(f->getIsEventSource() != f->getIsEventTarget())
      throw x3d::sai::InvalidAccessTypeException();
    const SequenceFieldBase *s =
      dynamic_cast<const SequenceFieldBase *>(f);
    if(!s) throw x3d::sai::InvalidAccessTypeException();
    Ref<ValueBase> w = im(v);
    if(w->getFieldType() != f->getType()->getSingleType())
      throw x3d::sai::InvalidValueException();
    handleChange(new FieldListChange(n, s, w, true));
  }
  catch(TypeMismatchException &)
  {
```

```
    throw x3d::sai::InvalidValueException();
  }
  catch(IllegalNodeIdException &)
  {
    throw x3d::sai::InvalidNodeException();
  }
  catch(Node::AccessException &)
  {
    // Note: This exception will not be utilized until we
    // implement a synchronized setting mechanism
    throw x3d::sai::InvalidOperationTimingException();
  }
}


void Session::handleChange(Ref<const ChangeBase> c)
{
  Mutex::Lock lock(mutex);
  if(buffered || !synchronizedChanges)
  {
    changeQueue.push(c);
    if(!buffered) executeChanges();
    return;
  }
  Time time = listener->server->getNextTimeStamp();
  c->apply(time);
}

void Session::handleExternalEvent(const ValueBase *v,
                                  unsigned long cookie)
{
  externalEventQueue->addEvent(v, cookie);
}

void Session::addRouteHead(Ref<const ExternalRouteHead> r)
{
  Mutex::Lock l(routeHeadsMutex);
  routeHeads.push_back(r);
}

void Session::removeRouteHead(Ref<const ExternalRouteHead> r)
{
  Mutex::Lock l(routeHeadsMutex);
  routeHeads.remove(r);
}



Session::ExternalEventQueue::ExternalEventQueue
(Ref<Session> session, Time heartbeat):
  session(session), heartbeat(heartbeat), nonEmpty(mutex),
  hasTerminated(false), flushRequest(false), flushing(false),
  flushed(mutex)
{
}

void Session::ExternalEventQueue::main()
{
```



```
  for(;;)
  {
    list<ExternalEvent> q;

    try
    {
      Mutex::Lock l(mutex);
      flushing = false;
      if(!flushRequest)
      {
        l.release();
        flushed.notifyAll();
        l.aquire(mutex);
      }
      while(eventQueue.empty())
        if(nonEmpty.timedWait(Time::now()+heartbeat)) break;
      std::swap(q, eventQueue);
      flushing = flushRequest;
      flushRequest = false;
    }
    catch(ThreadTerminatedException &) { break; }

    try { session->pushEvents(&q); }
    catch(ForwardDestroyedException &) { break; }
  }

  {
    Mutex::Lock l(mutex);
    hasTerminated = true;
    flushRequest = flushing = false;
  }
  flushed.notifyAll();
}

void Session::ExternalEventQueue::addEvent(const ValueBase *v,
                                           unsigned long cookie)
{
  Mutex::Lock l(mutex);
  if(hasTerminated) return;
  eventQueue.push_back(ExternalEvent(v, cookie));
  nonEmpty.notifyAll();
}

void Session::ExternalEventQueue::flush()
{
  Mutex::Lock l(mutex);
  if(hasTerminated) return;
  flushRequest = true;
  while(flushRequest || flushing) flushed.wait();
}


void Session::pushEvents(const list<ExternalEvent> *q)
{
  unsigned long n = q->size();
  x3d::sai::EventSeq s(n);
  s.length(n);
  unsigned j = 0;
```

```
      list<ExternalEvent>::const_iterator i = q->begin();
      while(i != q->end())
      {
        x3d::sai::Event &e = s[j++];
        e.cookie = i->cookie;
        ex(i->value.get(), e.val);
        ++i;
      }
      try { application->handleEvent(s); }
      catch(...)
      {
        disposeSession();
        return;
      }
    }


    Ref<Stream::Reader> Session::request(string p)
      throw(FileServer::RequestException)
    {
      x3d::sai::StreamReader_var r;
      try { r = application->requestFile(p.c_str()); }
      catch(x3d::sai::FileRequestException &e)
      {
        ARCHON_THROW1(FileServer::RequestException, string(e.message));
      }
      catch(...)
      {
        ARCHON_THROW1(FileServer::RequestException,
                      "Caught unknown exception");
      }
      if(CORBA::is_nil(r))
        ARCHON_THROW1(FileServer::RequestException,
                      "Got nil reader");
      return new Reader(r);
    }

    void Session::log(string s) throw()
    {
      try { application->log(s.c_str()); }
      catch(...) {}
    }
  }
 }
}
```

Listing: **archon/x3d/sai/value.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
```



```
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */
#ifndef ARCHON_X3D_SAI_VALUE_H
#define ARCHON_X3D_SAI_VALUE_H

#include <archon/x3d/server/field_type.H>
#include <archon/x3d/server/event.H>

#include <archon/x3d/sai/exception.H>
#include <archon/x3d/sai/sai.H>

namespace Archon
{
  namespace X3D
  {
    namespace SAI
    {
      using namespace Utilities;

      struct Porter
      {
        struct TypeMismatchException: ArgumentException
        {
          TypeMismatchException(string l): Utilities::Exception(l) {}
        };

        struct IllegalNodeIdException: ArgumentException
        {
          IllegalNodeIdException(string l): Utilities::Exception(l) {}
        };

        virtual unsigned long node2id(Ref<Node>) = 0;

        virtual Ref<Node> id2node(unsigned long)
          throw(IllegalNodeIdException) = 0;

        Ref<ValueBase> im(const x3d::sai::Value &v)
          throw(TypeMismatchException, IllegalNodeIdException)
        {
          unsigned type = v.type;
          if(type >= fieldTypeIndexRoof) ARCHON_THROW(TypeMismatchException);
          return (this->*importers[type])(v);
        }

        void ex(const ValueBase *source, x3d::sai::Value &target)
        {
          unsigned type = source->getFieldType()->getIndex();
          (this->*exporters[type])(source, target);
          target.type = type;
        }

      protected:
```

```
      Porter();
      virtual ~Porter() {}

    private:
      typedef Ref<ValueBase> (Porter::*Importer)
        (const x3d::sai::Value &);
      typedef void (Porter::*Exporter)
        (const ValueBase *source, x3d::sai::Value &target);

      vector<Importer> importers;
      vector<Exporter> exporters;

      template<typename F>
      Ref<ValueBase> imScalar(const x3d::sai::Value &);
      template<typename F>
      Ref<ValueBase> imTuple(const x3d::sai::Value &);
      template<typename F>
      Ref<ValueBase> imScalarSeq(const x3d::sai::Value &);
      template<typename F>
      Ref<ValueBase> imTupleSeq(const x3d::sai::Value &);

      Ref<ValueBase> imSFImage(const x3d::sai::Value &);
      Ref<ValueBase> imMFImage(const x3d::sai::Value &);

      Ref<ValueBase> imSFRotation(const x3d::sai::Value &);
      Ref<ValueBase> imMFRotation(const x3d::sai::Value &);

      Ref<ValueBase> imSFTime(const x3d::sai::Value &);
      Ref<ValueBase> imMFTime(const x3d::sai::Value &);


      template<typename F>
      void exScalar(const ValueBase *, x3d::sai::Value &);
      template<typename F>
      void exTuple(const ValueBase *, x3d::sai::Value &);
      template<typename F>
      void exScalarSeq(const ValueBase *, x3d::sai::Value &);
      template<typename F>
      void exTupleSeq(const ValueBase *, x3d::sai::Value &);

      void exSFImage(const ValueBase *, x3d::sai::Value &);
      void exMFImage(const ValueBase *, x3d::sai::Value &);

      void exSFRotation(const ValueBase *, x3d::sai::Value &);
      void exMFRotation(const ValueBase *, x3d::sai::Value &);

      void exSFTime(const ValueBase *, x3d::sai::Value &);
      void exMFTime(const ValueBase *, x3d::sai::Value &);
    };
  }
 }
}

#endif // ARCHON_X3D_SAI_VALUE_H
```

Listing: **archon/x3d/sai/value.C**

```
/*
```



```
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#include <archon/x3d/sai/value.H>

namespace Archon
{
  namespace X3D
  {
    namespace SAI
    {
      template<typename T> struct CorbaVal {};

      template<> struct CorbaVal<bool>
      {
        typedef CORBA::Boolean Type;
        typedef x3d::sai::booleanarray Array;
        typedef x3d::sai::booleanarray_var ArrayVar;
        static bool imp(Porter *, Type v) { return bool(v); }
        static Type exp(Porter *, bool v) { return Type(v); }
        static bool fromAny(const CORBA::Any &a, Type &v)
        { return a >>= CORBA::Any::to_boolean(v); }
        static void toAny(CORBA::Any &a, Type v)
        { a <<= CORBA::Any::from_boolean(v); }
      };

      template<> struct CorbaVal<int>
      {
        typedef CORBA::Long Type;
        typedef x3d::sai::longarray Array;
        typedef x3d::sai::longarray_var ArrayVar;
        typedef x3d::sai::longdoublearray DoubleArray;
        typedef x3d::sai::longdoublearray_var DoubleArrayVar;
        static int imp(Porter *, Type v) { return int(v); }
        static Type exp(Porter *, int v) { return Type(v); }
        static bool fromAny(const CORBA::Any &a, Type &v) { return a >>= v; }
        static void toAny(CORBA::Any &a, Type v) { a <<= v; }
      };

      template<> struct CorbaVal<long>
      {
        typedef CORBA::Long Type;
        typedef x3d::sai::longarray Array;
```

```
  typedef x3d::sai::longarray_var ArrayVar;
  typedef x3d::sai::longdoublearray DoubleArray;
  typedef x3d::sai::longdoublearray_var DoubleArrayVar;
  static long imp(Porter *, Type v) { return long(v); }
  static Type exp(Porter *, long v) { return Type(v); }
  static bool fromAny(const CORBA::Any &a, Type &v) { return a >>= v; }
  static void toAny(CORBA::Any &a, Type v) { a <<= v; }
};

template<> struct CorbaVal<float>
{
  typedef CORBA::Float Type;
  typedef x3d::sai::floatarray Array;
  typedef x3d::sai::floatarray_var ArrayVar;
  typedef x3d::sai::floatdoublearray DoubleArray;
  typedef x3d::sai::floatdoublearray_var DoubleArrayVar;
  static float imp(Porter *, Type v) { return float(v); }
  static Type exp(Porter *, float v) { return Type(v); }
  static bool fromAny(const CORBA::Any &a, Type &v) { return a >>= v; }
  static void toAny(CORBA::Any &a, Type v) { a <<= v; }
};

template<> struct CorbaVal<double>
{
  typedef CORBA::Double Type;
  typedef x3d::sai::doublearray Array;
  typedef x3d::sai::doublearray_var ArrayVar;
  typedef x3d::sai::doubledoublearray DoubleArray;
  typedef x3d::sai::doubledoublearray_var DoubleArrayVar;
  static double imp(Porter *, Type v) { return double(v); }
  static Type exp(Porter *, double v) { return Type(v); }
  static bool fromAny(const CORBA::Any &a, Type &v) { return a >>= v; }
  static void toAny(CORBA::Any &a, Type v) { a <<= v; }
};

template<> struct CorbaVal<string>
{
  typedef const char *Type;
  typedef x3d::sai::stringarray Array;
  typedef x3d::sai::stringarray_var ArrayVar;
  static string imp(Porter *, Type v) { return string(v); }
  static Type exp(Porter *, string v)
  { return CORBA::string_dup(v.c_str()); }
  static bool fromAny(const CORBA::Any &a, Type &v) { return a >>= v; }
  static void toAny(CORBA::Any &a, Type v) { a <<= v; }
};

template<> struct CorbaVal<Ref<Node> >
{
  typedef CORBA::ULong Type;
  typedef x3d::sai::ulongarray Array;
  typedef x3d::sai::ulongarray_var ArrayVar;
  static Ref<Node> imp(Porter *p, Type v)
  { return p->id2node(static_cast<unsigned long>(v)); }
  static Type exp(Porter *p, Ref<Node> v)
  { return Type(p->node2id(v)); }
  static bool fromAny(const CORBA::Any &a, Type &v) { return a >>= v; }
  static void toAny(CORBA::Any &a, Type v) { a <<= v; }
```



```
};



template<typename F>
Ref<ValueBase> Porter::imScalar(const x3d::sai::Value &v)
{
  typedef typename F::Type T;
  typedef typename F::Value V;
  typedef CorbaVal<T> C;
  typename C::Type w;
  if(!C::fromAny(v.val, w)) ARCHON_THROW(TypeMismatchException);
  return new V(F::type, C::imp(this, w));
}

template<typename F>
Ref<ValueBase> Porter::imTuple(const x3d::sai::Value &v)
{
  typedef typename F::Type T;
  typedef typename F::Value V;
  typedef typename F::ElemType E;
  typedef CorbaVal<E> C;
  const typename C::Array *w;
  if(!(v.val >>= w)) ARCHON_THROW(TypeMismatchException);
  if(w->length() != F::size) ARCHON_THROW(TypeMismatchException);
  Ref<V> r = new V(F::type);
  T &u = r->value;
  for(unsigned i=0; i<F::size; ++i) u[i] = C::imp(this, (*w)[i]);
  return r;
}

template<typename F>
Ref<ValueBase> Porter::imScalarSeq(const x3d::sai::Value &v)
{
  typedef typename F::Type T;
  typedef typename F::Value V;
  typedef typename F::Single::Type E;
  typedef CorbaVal<E> C;
  const typename C::Array *w;
  if(!(v.val >>= w)) ARCHON_THROW(TypeMismatchException);
  Ref<V> r = new V(F::type);
  T &u = r->value;
  u.resize(w->length());
  for(unsigned i=0; i<w->length(); ++i) u[i] = C::imp(this, (*w)[i]);
  return r;
}

template<typename F>
Ref<ValueBase> Porter::imTupleSeq(const x3d::sai::Value &v)
{
  typedef typename F::Type T;
  typedef typename F::Value V;
  typedef typename F::Single S;
  typedef typename S::ElemType E;
  typedef CorbaVal<E> C;
  const typename C::DoubleArray *w;
  if(!(v.val >>= w)) ARCHON_THROW(TypeMismatchException);
```

```
  Ref<V> r = new V(F::type);
  r->value.resize(w->length());
  for(unsigned i=0; i<w->length(); ++i)
  {
    typename S::Type &u = r->value[i];
    const typename C::Array &t = (*w)[i];
    if(t.length() != S::size) ARCHON_THROW(TypeMismatchException);
    for(unsigned j=0; j<S::size; ++j) u[j] = C::imp(this, t[j]);
  }
  return r;
}

/**
 * \todo Needs to be implemented
 */
Ref<ValueBase> Porter::imSFImage(const x3d::sai::Value &v)
{
  ARCHON_THROW1(InternalException, "imSFImage not yet implemented");
}

/**
 * \todo Needs to be implemented
 */
Ref<ValueBase> Porter::imMFImage(const x3d::sai::Value &v)
{
  ARCHON_THROW1(InternalException, "imMFImage not yet implemented");
}

Ref<ValueBase> Porter::imSFRotation(const x3d::sai::Value &v)
{
  typedef SFRotation F;
  typedef F::Type T;
  typedef F::Value V;
  typedef F::ElemType E;
  typedef CorbaVal<E> C;
  const C::Array *w;
  if(!(v.val >>= w)) ARCHON_THROW(TypeMismatchException);
  if(w->length() != F::size) ARCHON_THROW(TypeMismatchException);
  Ref<V> r = new V(F::type);
  T &u = r->value;
  u.normalizedAxis[0] = C::imp(this, (*w)[0]);
  u.normalizedAxis[1] = C::imp(this, (*w)[1]);
  u.normalizedAxis[2] = C::imp(this, (*w)[2]);
  u.angle = C::imp(this, (*w)[3]);
  return r;
}

Ref<ValueBase> Porter::imMFRotation(const x3d::sai::Value &v)
{
  typedef MFRotation F;
  typedef F::Type T;
  typedef F::Value V;
  typedef F::Single S;
  typedef S::ElemType E;
  typedef CorbaVal<E> C;
  const C::DoubleArray *w;
  if(!(v.val >>= w)) ARCHON_THROW(TypeMismatchException);
  Ref<V> r = new V(F::type);
```



```
  r->value.resize(w->length());
  for(unsigned i=0; i<w->length(); ++i)
  {
    S::Type &u = r->value[i];
    const C::Array &t = (*w)[i];
    if(t.length() != S::size) ARCHON_THROW(TypeMismatchException);
    u.normalizedAxis[0] = C::imp(this, t[0]);
    u.normalizedAxis[1] = C::imp(this, t[1]);
    u.normalizedAxis[2] = C::imp(this, t[2]);
    u.angle = C::imp(this, t[3]);
  }
  return r;
}

Ref<ValueBase> Porter::imSFTime(const x3d::sai::Value &v)
{
  typedef SFTime F;
  typedef F::Type T;
  typedef F::Value V;
  typedef F::ElemType E;
  typedef CorbaVal<E> C;
  const C::Array *w;
  if(!(v.val >>= w)) ARCHON_THROW(TypeMismatchException);
  if(w->length() != F::size) ARCHON_THROW(TypeMismatchException);
  Ref<V> r = new V(F::type);
  r->value.set(C::imp(this, (*w)[0]), C::imp(this, (*w)[1]));
  return r;
}

Ref<ValueBase> Porter::imMFTime(const x3d::sai::Value &v)
{
  typedef MFTime F;
  typedef F::Type T;
  typedef F::Value V;
  typedef F::Single S;
  typedef S::ElemType E;
  typedef CorbaVal<E> C;
  const C::DoubleArray *w;
  if(!(v.val >>= w)) ARCHON_THROW(TypeMismatchException);
  Ref<V> r = new V(F::type);
  r->value.resize(w->length());
  for(unsigned i=0; i<w->length(); ++i)
  {
    const C::Array &t = (*w)[i];
    if(t.length() != S::size) ARCHON_THROW(TypeMismatchException);
    r->value[i].set(C::imp(this, t[0]), C::imp(this, t[1]));
  }
  return r;
}



template<typename F>
void Porter::exScalar(const ValueBase *s, x3d::sai::Value &t)
{
  typedef typename F::Type T;
  typedef typename F::Value V;
  typedef CorbaVal<T> C;
```

```
  const V *v = dynamic_cast<const V *>(s);
  if(!v) ARCHON_THROW1(InternalException, "Type mismatch");
  C::toAny(t.val, C::exp(this, v->value));
}

template<typename F>
void Porter::exTuple(const ValueBase *s, x3d::sai::Value &t)
{
  typedef typename F::Type T;
  typedef typename F::Value V;
  typedef typename F::ElemType E;
  typedef CorbaVal<E> C;
  const V *v = dynamic_cast<const V *>(s);
  if(!v) ARCHON_THROW1(InternalException, "Type mismatch");
  const T &u = v->value;
  typename C::ArrayVar w(new typename C::Array(F::size));
  w->length(F::size);
  for(unsigned i=0; i<F::size; ++i) w[i] = C::exp(this, u[i]);
  t.val <<= w._retn();
}

template<typename F>
void Porter::exScalarSeq(const ValueBase *s, x3d::sai::Value &t)
{
  typedef typename F::Type T;
  typedef typename F::Value V;
  typedef typename F::Single::Type E;
  typedef CorbaVal<E> C;
  const V *v = dynamic_cast<const V *>(s);
  if(!v) ARCHON_THROW1(InternalException, "Type mismatch");
  typename C::ArrayVar w(new typename C::Array(v->value.size()));
  w->length(v->value.size());
  for(unsigned i=0; i<v->value.size(); ++i)
    w[i] = C::exp(this, v->value[i]);
  t.val <<= w._retn();
}

template<typename F>
void Porter::exTupleSeq(const ValueBase *s, x3d::sai::Value &t)
{
  typedef typename F::Type T;
  typedef typename F::Value V;
  typedef typename F::Single S;
  typedef typename S::ElemType E;
  typedef CorbaVal<E> C;
  const V *v = dynamic_cast<const V *>(s);
  if(!v) ARCHON_THROW1(InternalException, "Type mismatch");
  typename C::DoubleArrayVar
    w(new typename C::DoubleArray(v->value.size()));
  w->length(v->value.size());
  for(unsigned i=0; i<v->value.size(); ++i)
  {
    typename C::Array &u = w[i];
    u.length(S::size);
    const typename S::Type &x = v->value[i];
    for(unsigned j=0; j<S::size; ++j) u[j] = C::exp(this, x[j]);
  }
  t.val <<= w._retn();
```



```
}

/**
 * \todo Needs to be implemented
 */
void Porter::exSFImage(const ValueBase *s, x3d::sai::Value &t)
{
  ARCHON_THROW1(InternalException, "exSFImage not yet implemented");
}

/**
 * \todo Needs to be implemented
 */
void Porter::exMFImage(const ValueBase *s, x3d::sai::Value &t)
{
  ARCHON_THROW1(InternalException, "exMFImage not yet implemented");
}

void Porter::exSFRotation(const ValueBase *s, x3d::sai::Value &t)
{
  typedef SFRotation F;
  typedef F::Type T;
  typedef F::Value V;
  typedef F::ElemType E;
  typedef CorbaVal<E> C;
  const V *v = dynamic_cast<const V *>(s);
  if(!v) ARCHON_THROW1(InternalException, "Type mismatch");
  const T &u = v->value;
  C::ArrayVar w(new C::Array(F::size));
  w->length(F::size);
  w[0] = C::exp(this, u.normalizedAxis[0]);
  w[1] = C::exp(this, u.normalizedAxis[1]);
  w[2] = C::exp(this, u.normalizedAxis[2]);
  w[3] = C::exp(this, u.angle);
  t.val <<= w._retn();
}

void Porter::exMFRotation(const ValueBase *s, x3d::sai::Value &t)
{
  typedef MFRotation F;
  typedef F::Type T;
  typedef F::Value V;
  typedef F::Single S;
  typedef S::ElemType E;
  typedef CorbaVal<E> C;
  const V *v = dynamic_cast<const V *>(s);
  if(!v) ARCHON_THROW1(InternalException, "Type mismatch");
  C::DoubleArrayVar w(new C::DoubleArray(v->value.size()));
  w->length(v->value.size());
  for(unsigned i=0; i<v->value.size(); ++i)
  {
    C::Array &u = w[i];
    u.length(S::size);
    const S::Type &x = v->value[i];
    u[0] = C::exp(this, x.normalizedAxis[0]);
    u[1] = C::exp(this, x.normalizedAxis[1]);
    u[2] = C::exp(this, x.normalizedAxis[2]);
    u[3] = C::exp(this, x.angle);
```

```
  }
  t.val <<= w._retn();
}

void Porter::exSFTime(const ValueBase *s, x3d::sai::Value &t)
{
  typedef SFTime F;
  typedef F::Type T;
  typedef F::Value V;
  typedef F::ElemType E;
  typedef CorbaVal<E> C;
  const V *v = dynamic_cast<const V *>(s);
  if(!v) ARCHON_THROW1(InternalException, "Type mismatch");
  C::ArrayVar w(new C::Array(F::size));
  w->length(F::size);
  long sec;
  long nsec;
  v->value.get(sec, nsec);
  w[0] = C::exp(this, sec);
  w[1] = C::exp(this, nsec);
  t.val <<= w._retn();
}

void Porter::exMFTime(const ValueBase *s, x3d::sai::Value &t)
{
  typedef MFTime F;
  typedef F::Type T;
  typedef F::Value V;
  typedef F::Single S;
  typedef S::ElemType E;
  typedef CorbaVal<E> C;
  const V *v = dynamic_cast<const V *>(s);
  if(!v) ARCHON_THROW1(InternalException, "Type mismatch");
  C::DoubleArrayVar w(new C::DoubleArray(v->value.size()));
  w->length(v->value.size());
  for(unsigned i=0; i<v->value.size(); ++i)
  {
    C::Array &u = w[i];
    u.length(S::size);
    long sec;
    long nsec;
    v->value[i].get(sec, nsec);
    u[0] = C::exp(this, sec);
    u[1] = C::exp(this, nsec);
  }
  t.val <<= w._retn();
}

Porter::Porter()
{
  importers.resize(fieldTypeIndexRoof);
  exporters.resize(fieldTypeIndexRoof);

  exporters[SFBool::index] = &Porter::exScalar<SFBool>;
  importers[SFBool::index] = &Porter::imScalar<SFBool>;
  exporters[MFBool::index] = &Porter::exScalarSeq<MFBool>;
  importers[MFBool::index] = &Porter::imScalarSeq<MFBool>;
```



```
  exporters[SFColor::index] = &Porter::exTuple<SFColor>;
  importers[SFColor::index] = &Porter::imTuple<SFColor>;
  exporters[MFColor::index] = &Porter::exTupleSeq<MFColor>;
  importers[MFColor::index] = &Porter::imTupleSeq<MFColor>;

  exporters[SFColorRGBA::index] = &Porter::exTuple<SFColorRGBA>;
  importers[SFColorRGBA::index] = &Porter::imTuple<SFColorRGBA>;
  exporters[MFColorRGBA::index] = &Porter::exTupleSeq<MFColorRGBA>;
  importers[MFColorRGBA::index] = &Porter::imTupleSeq<MFColorRGBA>;

  exporters[SFDouble::index] = &Porter::exScalar<SFDouble>;
  importers[SFDouble::index] = &Porter::imScalar<SFDouble>;
  exporters[MFDouble::index] = &Porter::exScalarSeq<MFDouble>;
  importers[MFDouble::index] = &Porter::imScalarSeq<MFDouble>;

  exporters[SFFloat::index] = &Porter::exScalar<SFFloat>;
  importers[SFFloat::index] = &Porter::imScalar<SFFloat>;
  exporters[MFFloat::index] = &Porter::exScalarSeq<MFFloat>;
  importers[MFFloat::index] = &Porter::imScalarSeq<MFFloat>;

  exporters[SFImage::index] = &Porter::exSFImage;
  importers[SFImage::index] = &Porter::imSFImage;
  exporters[MFImage::index] = &Porter::exMFImage;
  importers[MFImage::index] = &Porter::imMFImage;

  exporters[SFInt32::index] = &Porter::exScalar<SFInt32>;
  importers[SFInt32::index] = &Porter::imScalar<SFInt32>;
  exporters[MFInt32::index] = &Porter::exScalarSeq<MFInt32>;
  importers[MFInt32::index] = &Porter::imScalarSeq<MFInt32>;

  exporters[SFNode::index] = &Porter::exScalar<SFNodeVar<Node> >;
  importers[SFNode::index] = &Porter::imScalar<SFNodeVar<Node> >;
  exporters[MFNode::index] = &Porter::exScalarSeq<MFNodeVar<Node> >;
  importers[MFNode::index] = &Porter::imScalarSeq<MFNodeVar<Node> >;

  exporters[SFRotation::index] = &Porter::exSFRotation;
  importers[SFRotation::index] = &Porter::imSFRotation;
  exporters[MFRotation::index] = &Porter::exMFRotation;
  importers[MFRotation::index] = &Porter::imMFRotation;

  exporters[SFString::index] = &Porter::exScalar<SFString>;
  importers[SFString::index] = &Porter::imScalar<SFString>;
  exporters[MFString::index] = &Porter::exScalarSeq<MFString>;
  importers[MFString::index] = &Porter::imScalarSeq<MFString>;

  exporters[SFTime::index] = &Porter::exSFTime;
  importers[SFTime::index] = &Porter::imSFTime;
  exporters[MFTime::index] = &Porter::exMFTime;
  importers[MFTime::index] = &Porter::imMFTime;

  exporters[SFVec2d::index] = &Porter::exTuple<SFVec2d>;
  importers[SFVec2d::index] = &Porter::imTuple<SFVec2d>;
  exporters[MFVec2d::index] = &Porter::exTupleSeq<MFVec2d>;
  importers[MFVec2d::index] = &Porter::imTupleSeq<MFVec2d>;

  exporters[SFVec2f::index] = &Porter::exTuple<SFVec2f>;
  importers[SFVec2f::index] = &Porter::imTuple<SFVec2f>;
  exporters[MFVec2f::index] = &Porter::exTupleSeq<MFVec2f>;
```

```
        importers[MFVec2f::index] = &Porter::imTupleSeq<MFVec2f>;

        exporters[SFVec3d::index] = &Porter::exTuple<SFVec3d>;
        importers[SFVec3d::index] = &Porter::imTuple<SFVec3d>;
        exporters[MFVec3d::index] = &Porter::exTupleSeq<MFVec3d>;
        importers[MFVec3d::index] = &Porter::imTupleSeq<MFVec3d>;

        exporters[SFVec3f::index] = &Porter::exTuple<SFVec3f>;
        importers[SFVec3f::index] = &Porter::imTuple<SFVec3f>;
        exporters[MFVec3f::index] = &Porter::exTupleSeq<MFVec3f>;
        importers[MFVec3f::index] = &Porter::imTupleSeq<MFVec3f>;
      }
    }
  }
}
```

Listing: **archon/x3d/server/appear.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_APPEAR_H
#define ARCHON_X3D_APPEAR_H

/*
 * \file appear.C
 *
 * X3D Appearance component
 *
 * NOTE: "X3DAppearanceChildNode" had to be moved from the Appearance
 * component to the Texture component to satisfy both inheritance and
 * containment hierachies.
 */

#include <archon/x3d/server/texture.H>

namespace Archon
{
  namespace X3D
  {
    struct MaterialNode: AppearanceChildNode
    {
```



```
      static const NodeType *type;

    protected:
      MaterialNode() {}
    };


    struct Material: MaterialNode
    {
      static Ref<Node> instantiate(BackRef<ExecutionContext> c)
      {
        return new Material(c);
      }

      static const NodeType *type;
      const NodeType *getType() const { return type; }

      Material(BackRef<ExecutionContext> c):
        Node(c),
        diffuseColor(0.8, 0.8, 0.8),
        specularColor(0, 0, 0),
        emissiveColor(0, 0, 0),
        ambientIntensity(0.2),
        shininess(0.2),
        transparency(0),
        refractiveIndex(1) {}

      const Vector3 &getDiffuseColor()  const { return diffuseColor; }
      const Vector3 &getSpecularColor() const { return specularColor; }
      const Vector3 &getEmissiveColor() const { return emissiveColor; }
      double getAmbientIntensity() const { return ambientIntensity; }
      double getShininess()        const { return shininess; }
      double getTransparency()     const { return transparency; }
      double getRefractiveIndex()  const { return refractiveIndex; }

    private:
      friend void initializeAppearComponent();

      Vector3 diffuseColor;
      Vector3 specularColor;
      Vector3 emissiveColor;
      double ambientIntensity;
      double shininess;       // Relates to the Phong exponent
      double transparency;
      double refractiveIndex; // Not part of WRML or X3D

      EventSource diffuseColorChanged;
      EventSource specularColorChanged;
      EventSource emissiveColorChanged;
      EventSource ambientIntensityChanged;
      EventSource shininessChanged;
      EventSource transparencyChanged;
      EventSource refractiveIndexChanged;

      Time diffuseColorStamp;
      Time specularColorStamp;
      Time emissiveColorStamp;
      Time ambientIntensityStamp;
```

```
  Time shininessStamp;
  Time transparencyStamp;
  Time refractiveIndexStamp;
};


struct AppearanceNode: virtual Node
{
  static const NodeType *type;

protected:
  AppearanceNode() {}
};


struct Appearance: AppearanceNode
{
  static Ref<Node> instantiate(BackRef<ExecutionContext> c)
  {
    return new Appearance(c);
  }

  static const NodeType *type;
  const NodeType *getType() const { return type; }

  Appearance(BackRef<ExecutionContext> c): Node(c) {}

  virtual ~Appearance() {}

  Ref<const MaterialNode> getMaterial() const { return material; }
  Ref<      MaterialNode> getMaterial()       { return material; }
  Ref<const TextureNode>  getTexture()  const { return texture; }
  Ref<      TextureNode>  getTexture()        { return texture; }
  Ref<const TextureTransformNode> getTextureTransform() const { return
       textureTransform; }
  Ref<      TextureTransformNode> getTextureTransform()       { return
       textureTransform; }

private:
  friend void initializeAppearComponent();

  Ref<MaterialNode>         material;
  Ref<TextureNode>          texture;
  Ref<TextureTransformNode> textureTransform;

  EventSource materialChanged;
  EventSource textureChanged;
  EventSource textureTransformChanged;

  Time materialStamp;
  Time textureStamp;
  Time textureTransformStamp;
};


struct BindableNode: virtual ChildNode
{
  static const NodeType *type;
```



```
protected:
  BindableNode(): isBound(false) {}

private:
  friend void initializeAppearComponent();

  bool isBound;
  Time bindTime;

  /**
   * \todo What are these??
   */
  void setBind(const bool &)
  {
    ARCHON_THROW1(InternalException, "Background.set_bind is not
         supported yet");
  }

  EventSource isBoundChanged;
  EventSource bindTimeChanged;

  Time bindStamp;
};


struct Background: BindableNode
{
  static Ref<Node> instantiate(BackRef<ExecutionContext> c)
  {
    return new Background(c);
  }

  static const NodeType *type;
  const NodeType *getType() const { return type; }

  Background(BackRef<ExecutionContext> c): Node(c)
  {
    skyColor.push_back(Vector3(0, 0, 0));
  }

private:
  friend void initializeAppearComponent();

  vector<double>  groundAngle;
  vector<Vector3> groundColor;
  vector<string>  backUrl;
  vector<string>  bottomUrl;
  vector<string>  frontUrl;
  vector<string>  leftUrl;
  vector<string>  rightUrl;
  vector<string>  topUrl;
  vector<double>  skyAngle;
  vector<Vector3> skyColor;

  EventSource groundAngleChanged;
  EventSource groundColorChanged;
  EventSource backUrlChanged;
```

```
      EventSource bottomUrlChanged;
      EventSource frontUrlChanged;
      EventSource leftUrlChanged;
      EventSource rightUrlChanged;
      EventSource topUrlChanged;
      EventSource skyAngleChanged;
      EventSource skyColorChanged;

      Time groundAngleStamp;
      Time groundColorStamp;
      Time backUrlStamp;
      Time bottomUrlStamp;
      Time frontUrlStamp;
      Time leftUrlStamp;
      Time rightUrlStamp;
      Time topUrlStamp;
      Time skyAngleStamp;
      Time skyColorStamp;
    };


    struct Fog: BindableNode
    {
      static Ref<Node> instantiate(BackRef<ExecutionContext> c)
      {
        return new Fog(c);
      }

      static const NodeType *type;
      const NodeType *getType() const { return type; }

      Fog(BackRef<ExecutionContext> c):
        Node(c), color(1, 1, 1), fogType("LINEAR"),
        visibilityRange(0) {}

    private:
      friend void initializeAppearComponent();

      Vector3 color;
      string fogType;
      double visibilityRange;

      EventSource colorChanged;
      EventSource fogTypeChanged;
      EventSource visibilityRangeChanged;

      Time colorStamp;
      Time fogTypeStamp;
      Time visibilityRangeStamp;
    };
  }
}

#endif // ARCHON_X3D_APPEAR_H
```

Listing: **archon/x3d/server/appear.C**


```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

/*
 * \file appear.C
 *
 * X3D Appearance component
 *
 * NOTE: "X3DAppearanceChildNode" had to be moved from the Appearance
 * component to the Texture component to satisfy both inheritance and
 * containment hierachies.
 */

#include <archon/x3d/server/field_type.H>
#include <archon/x3d/server/field.H>
#include <archon/x3d/server/appear.H>

namespace Archon
{
  namespace X3D
  {
    extern long nodeTypeIndexEnumerator;

    const NodeType *MaterialNode::type = 0;
    const NodeType *Material::type = 0;

    const NodeType *AppearanceNode::type = 0;
    const NodeType *Appearance::type = 0;

    const NodeType *BindableNode::type = 0;
    const NodeType *Fog::type = 0;
    const NodeType *Background::type = 0;

    void initializeAppearComponent()
    {
      vector<const FieldBase *> fields;


      // MaterialNode

      MaterialNode::type =
        NodeType::newAbstract("MaterialNode", false, 0,
                              AppearanceChildNode::type);
```

```
    // Material

    fields.push_back(newExposedField("ambientIntensity", SFFloat::type,
                                     &Material::ambientIntensity,
                                     &Material::ambientIntensityChanged,
                                     &Material::ambientIntensityStamp));
    fields.push_back(newExposedField("diffuseColor", SFColor::type,
                                     &Material::diffuseColor,
                                     &Material::diffuseColorChanged,
                                     &Material::diffuseColorStamp));
    fields.push_back(newExposedField("emissiveColor", SFColor::type,
                                     &Material::emissiveColor,
                                     &Material::emissiveColorChanged,
                                     &Material::emissiveColorStamp));
    fields.push_back(newExposedField("shininess", SFFloat::type,
                                     &Material::shininess,
                                     &Material::shininessChanged,
                                     &Material::shininessStamp));
    fields.push_back(newExposedField("specularColor", SFColor::type,
                                     &Material::specularColor,
                                     &Material::specularColorChanged,
                                     &Material::specularColorStamp));
    fields.push_back(newExposedField("transparency", SFFloat::type,
                                     &Material::transparency,
                                     &Material::transparencyChanged,
                                     &Material::transparencyStamp));
    fields.push_back(newExposedField("refractiveIndex", SFFloat::type,
                                     &Material::refractiveIndex,
                                     &Material::refractiveIndexChanged,
                                     &Material::refractiveIndexStamp));
    Material::type =
      NodeType::newConcrete("Material", Material::instantiate, &fields,
                            MaterialNode::type);
    fields.clear();


    // AppearanceNode

    AppearanceNode::type =
      NodeType::newAbstract("AppearanceNode", false, 0,
                            Node::type);


    // Appearance

    fields.push_back(newExposedField("material",
                                     &Appearance::material,
                                     &Appearance::materialChanged,
                                     &Appearance::materialStamp));
    fields.push_back(newExposedField("texture",
                                     &Appearance::texture,
                                     &Appearance::textureChanged,
                                     &Appearance::textureStamp));
    fields.push_back(newExposedField("textureTransform",
                                     &Appearance::textureTransform,
```



```
                                     &Appearance::textureTransformChanged,
                                     &Appearance::textureTransformStamp));
    Appearance::type =
      NodeType::newConcrete("Appearance", Appearance::instantiate, &fields,
                            AppearanceNode::type);
    fields.clear();


    // BindableNode

    fields.push_back(newEventIn("bind", SFBool::type,
                                &BindableNode::setBind,
                                &BindableNode::bindStamp));
    fields.push_back(newEventOut("isBound", SFBool::type,
                                 &BindableNode::isBound,
                                 &BindableNode::isBoundChanged));
    fields.push_back(newEventOut("bindTime", SFTime::type,
                                 &BindableNode::bindTime,
                                 &BindableNode::bindTimeChanged));
    BindableNode::type =
      NodeType::newAbstract("BindableNode", false, &fields,
                            ChildNode::type);
    fields.clear();


    // Fog

    /*
     *  Note: 'set_bind' should have been an EventIn field - it is in
     *  the VRML97 specification.
     */
    fields.push_back(newExposedField("color", SFColor::type,
                                     &Fog::color,
                                     &Fog::colorChanged,
                                     &Fog::colorStamp));
    fields.push_back(newExposedField("fogType", SFString::type,
                                     &Fog::fogType,
                                     &Fog::fogTypeChanged,
                                     &Fog::fogTypeStamp));
    fields.push_back(newExposedField("visibilityRange", SFFloat::type,
                                     &Fog::visibilityRange,
                                     &Fog::visibilityRangeChanged,
                                     &Fog::visibilityRangeStamp));
    Fog::type =
      NodeType::newConcrete("Fog", Fog::instantiate, &fields,
                            BindableNode::type);
    fields.clear();


    // Background

    fields.push_back(newExposedField("groundAngle", MFFloat::type,
                                     &Background::groundAngle,
                                     &Background::groundAngleChanged,
                                     &Background::groundAngleStamp));
    fields.push_back(newExposedField("groundColor", MFVec3f::type,
                                     &Background::groundColor,
```

```
                                    &Background::groundColorChanged,
                                    &Background::groundColorStamp));
      fields.push_back(newExposedField("backUrl", MFString::type,
                                    &Background::backUrl,
                                    &Background::backUrlChanged,
                                    &Background::backUrlStamp));
      fields.push_back(newExposedField("bottomUrl", MFString::type,
                                    &Background::bottomUrl,
                                    &Background::bottomUrlChanged,
                                    &Background::bottomUrlStamp));
      fields.push_back(newExposedField("frontUrl", MFString::type,
                                    &Background::frontUrl,
                                    &Background::frontUrlChanged,
                                    &Background::frontUrlStamp));
      fields.push_back(newExposedField("leftUrl", MFString::type,
                                    &Background::leftUrl,
                                    &Background::leftUrlChanged,
                                    &Background::leftUrlStamp));
      fields.push_back(newExposedField("rightUrl", MFString::type,
                                    &Background::rightUrl,
                                    &Background::rightUrlChanged,
                                    &Background::rightUrlStamp));
      fields.push_back(newExposedField("topUrl", MFString::type,
                                    &Background::topUrl,
                                    &Background::topUrlChanged,
                                    &Background::topUrlStamp));
      fields.push_back(newExposedField("skyAngle", MFFloat::type,
                                    &Background::skyAngle,
                                    &Background::skyAngleChanged,
                                    &Background::skyAngleStamp));
      fields.push_back(newExposedField("skyColor", MFVec3f::type,
                                    &Background::skyColor,
                                    &Background::skyColorChanged,
                                    &Background::skyColorStamp));
      Background::type =
        NodeType::newConcrete("Background", Background::instantiate, &fields,
                              BindableNode::type);
      fields.clear();
    }
  }
}
```

Listing: **archon/x3d/server/aux.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
```



```
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

/*
 * \file aux.H
 *
 *  Auxiliary nodes component. Nodes which are not part of the X3D
 *  specification
 */

#ifndef ARCHON_X3D_AUX_H
#define ARCHON_X3D_AUX_H

#include <archon/x3d/server/geometry.H>

namespace Archon
{
  namespace X3D
  {
    /**
     * A torus geometry node.
     *
     * A torus is described by a major and a minor radius. The major
     * radius describes a circle in the z-x-plane centered at the
     * origin of the local coordinate system. Any point on the surface
     * of the torus can then be describes by a vector drawn from some
     * point on this circle. Each such vector must be perpendicular to
     * the circle at that point and be as long as the minor radius.
     * Thus, the torus is centered at the origin of the local
     * coordinate system and has the y-axis as the axis of revolution.
     * When textures are applied to a torus the texture x-coordinate
     * is taken as the angle of the projection of the surface point
     * onto the z-x-plane taking zero to be the direction of the
     * negative z-axis and increasing counter-clockwise when seen from
     * the positive y-axis. The texture y-coordinate is taken as the
     * angle between the surface point and the center of the torus
     * when seen from the point on the circle of revolution that is
     * closest to the surface point. When seen from a point on the
     * circle of revolution in the direction of revolution the texture
     * y-coordinate increses clockwise.
     */
    struct Torus: Geometry3DNode
    {
      static Ref<Node> instantiate(BackRef<ExecutionContext> c)
      {
        return new Torus(c);
      }

      static const NodeType *type;
      const NodeType *getType() const { return type; }

      Torus(BackRef<ExecutionContext> c):
        Node(c), majorRadius(1), minorRadius(0.5) {}

      double getMajorRadius() const { return majorRadius; }
      double getMinorRadius() const { return minorRadius; }
```

```
      int intersect(const Math::Ray3 &, double &dist) const;
      void render(bool texture, bool color,
                  const Shape *shape,
                  const RenderConfig *) const;

    private:
      friend void initializeAuxComponent();

      double majorRadius;
      double minorRadius;
    };
  }
}

#endif // ARCHON_X3D_AUX_H
```

Listing: **archon/x3d/server/aux.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

/*
 * \file aux.C
 *
 *  Auxiliary nodes component. Nodes which are not part of the X3D
 *  specification
 */

#include <GL/gl.h>

#include <archon/x3d/server/field_type.H>
#include <archon/x3d/server/field.H>
#include <archon/x3d/server/aux.H>

namespace Archon
{
  namespace X3D
  {
    extern long nodeTypeIndexEnumerator;

    const NodeType *Torus::type = 0;  // Not a part of X3D
```



```
    void initializeAuxComponent()
    {
      vector<const FieldBase *> fields;

      // Torus

      fields.push_back(newPrivateField("majorRadius", SFFloat::type,
                                       &Torus::majorRadius));
      fields.push_back(newPrivateField("minorRadius", SFFloat::type,
                                       &Torus::minorRadius));
      Torus::type =
        NodeType::newConcrete("Torus", Torus::instantiate, &fields,
                              Geometry3DNode::type);
      fields.clear();
    }

    int Torus::intersect(const Math::Ray3 &ray, double &dist) const
    {
      return Math::intersectTorus(ray, majorRadius, minorRadius, dist
           ) ? 1 : 0;
    }

    void renderNormals(const vector<Vector3> &, const RenderConfig *);

    void Torus::render(bool texture, bool color,
                       const Shape *shape,
                       const RenderConfig *config) const
    {
      glFrontFace(GL_CW);
      glDisable(GL_COLOR_MATERIAL);
      glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, 0);
      glEnable(GL_CULL_FACE);

      vector<Vector3> normals;
      const bool showNormals = config->showNormals;

      const int phiSteps = config->subdivision2;  // Around a section of the
           torus tube
      const int thetaSteps = config->subdivision1; // Around the hole

      vector<Vector3> nl(thetaSteps);
      vector<Vector3> vl(thetaSteps);
      vector<Vector3> fnl(thetaSteps);
      vector<Vector3> fvl(thetaSteps);

      for(int i=0; i<thetaSteps; ++i)
      {
        double texS = 1.0 / thetaSteps * i;
        double theta = 2 * M_PI * texS;

        double cos_theta = cos(theta);
        double sin_theta = sin(theta);

        fnl[i] = nl[i] = Vector3(sin_theta, 0, cos_theta);
        fvl[i] = vl[i] = nl[i] * (minorRadius - majorRadius);
      }
```

```
double prevTexT = 0;
Vector3 v;

for(int i=1; i<phiSteps; ++i)
{
  glBegin(GL_QUAD_STRIP);

  const Vector3 pvl0 = vl[0];
  const Vector3 pnl0 = nl[0];

  double texT = 1.0 / phiSteps * i;
  double phi = 2 * M_PI * texT;

  double cos_phi = cos(phi);
  double sin_phi = sin(phi);

  double a = majorRadius - minorRadius * cos_phi;
  double b = -minorRadius * sin_phi;

  for(int j=0; j<thetaSteps; ++j)
  {
    double texS = 1.0 / thetaSteps * j;
    double theta = 2 * M_PI * texS;

    double cos_theta = cos(theta);
    double sin_theta = sin(theta);

    if(texture) glTexCoord2d(texS, prevTexT);
    v = nl[j];
    glNormal3d(v[0], v[1], v[2]);
    v = vl[j];
    glVertex3d(v[0], v[1], v[2]);
    if(showNormals)
    {
      normals.push_back(v);
      normals.push_back(nl[j]);
    }

    if(texture) glTexCoord2d(texS, texT);
    v = nl[j] = Vector3(cos_phi*sin_theta, -sin_phi, cos_phi*cos_theta
         );
    glNormal3d(v[0], v[1], v[2]);
    v = vl[j] = Vector3(-sin_theta * a, b, -cos_theta * a);
    glVertex3d(v[0], v[1], v[2]);
  }

  if(texture) glTexCoord2d(1, prevTexT);
  glNormal3d(pnl0[0], pnl0[1], pnl0[2]);
  glVertex3d(pvl0[0], pvl0[1], pvl0[2]);

  if(texture) glTexCoord2d(1, texT);
  v = nl[0];
  glNormal3d(v[0], v[1], v[2]);
  v = vl[0];
  glVertex3d(v[0], v[1], v[2]);

  prevTexT = texT;
```



```
      glEnd();
    }

    glBegin(GL_QUAD_STRIP);

    for(int i=0; i<thetaSteps; ++i)
    {
      double texS = 1.0 / thetaSteps * i;

      if(texture) glTexCoord2d(texS, prevTexT);
      v = nl[i];
      glNormal3d(v[0], v[1], v[2]);
      v = vl[i];
      glVertex3d(v[0], v[1], v[2]);
      if(showNormals)
      {
        normals.push_back(v);
        normals.push_back(nl[i]);
      }

      if(texture) glTexCoord2d(texS, 1);
      v = fnl[i];
      glNormal3d(v[0], v[1], v[2]);
      v = fvl[i];
      glVertex3d(v[0], v[1], v[2]);
    }

    if(texture) glTexCoord2d(1, prevTexT);
    v = nl[0];
    glNormal3d(v[0], v[1], v[2]);
    v = vl[0];
    glVertex3d(v[0], v[1], v[2]);

    if(texture) glTexCoord2d(1, 1);
    v = fnl[0];
    glNormal3d(v[0], v[1], v[2]);
    v = fvl[0];
    glVertex3d(v[0], v[1], v[2]);

    glEnd();

    if(showNormals) renderNormals(normals, config);
  }
 }
}
```

Listing: **archon/x3d/server/core.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
```

```
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_CORE_H
#define ARCHON_X3D_CORE_H

/**
 * \file core.H
 *
 * X3D Core component
 */

#include <archon/x3d/server/type.H>
#include <archon/x3d/server/node.H>
#include <archon/x3d/server/event.H>

namespace Archon
{
  namespace X3D
  {
    class Viewer; // "view.H"


    struct ChildNode: virtual Node
    {
      static const NodeType *type;

    protected:
      ChildNode() {}
    };


    struct SensorNode: virtual ChildNode
    {
      static const NodeType *type;

      bool getEnabled() const { return enabled; }

    protected:
      SensorNode(): enabled(true), isActive(false) {}
      SensorNode(bool enabled):
        enabled(enabled), isActive(false) {}

      friend class Viewer;
      friend void initializeCoreComponent();

      bool enabled;
      bool isActive;

      EventSource enabledChanged;
      EventSource isActiveChanged;
```



```
      Time enabledStamp;
    };
  }
}

#endif // ARCHON_X3D_CORE_H
```

Listing: **archon/x3d/server/core.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

/**
 * \file core.C
 *
 * X3D Core component
 */

#include <archon/x3d/server/field_type.H>
#include <archon/x3d/server/field.H>
#include <archon/x3d/server/core.H>

namespace Archon
{
  namespace X3D
  {
    extern long nodeTypeIndexEnumerator;

    const NodeType *ChildNode::type = 0;
    const NodeType *SensorNode::type = 0;

    void initializeCoreComponent()
    {
      vector<const FieldBase *> fields;


      // ChildNode

      ChildNode::type =
        NodeType::newAbstract("ChildNode", true, 0, Node::type);
```

```
      // SensorNode

      fields.push_back(newExposedField("enabled", SFBool::type,
                                       &SensorNode::enabled,
                                       &SensorNode::enabledChanged,
                                       &SensorNode::enabledStamp));
      fields.push_back(newEventOut("isActive", SFBool::type,
                                   &SensorNode::isActive,
                                   &SensorNode::isActiveChanged));
      SensorNode::type =
        NodeType::newAbstract("SensorNode", false,
                              &fields, ChildNode::type);
      fields.clear();
    }
  }
}
```

Listing: **archon/x3d/server/custom_field.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_CUSTOM_FIELD_H
#define ARCHON_X3D_CUSTOM_FIELD_H

/*
 * \file custom_field.H
 *
 *  Definition of custom fields as used in scripts.
 */

#include <archon/x3d/server/field_type.H>
#include <archon/x3d/server/dump.H>
#include <archon/x3d/server/type.H>
#include <archon/x3d/server/event.H>
#include <archon/x3d/server/node.H>

namespace Archon
{
  namespace X3D
  {
    using namespace Archon::Math;
```



```
    struct Parser;
    struct CustomFieldBase;

    /**
     * The base of all nodes that can contain custom fields. That is,
     * fields which are user specified and thus not a part of the
     * static node type. An obvious example is the Script node.
     *
     * We should think of an unrealized CustomFieldNode as one where
     * we can still add new custom fields. Likewise we should think of
     * a realized CustomFieldNode as one where adding a new custom
     * field is an illegal operation.
     */
    struct CustomFieldNode: virtual Node
    {
      /**
       * Return false if the field name is in use.
       * May not be called for a realized node.
       */
      bool addCustomField(Ref<CustomFieldBase>) throw(RealizedException);

      /**
       * The 'customFields' variable is constant after realization of
       * the node. Therefor you can call this method and
       * 'getCustomField' without previously aquiring the scenegraph
       * mutex if you know that the node is realized.
       */
      unsigned getNumberOfCustomFields() const
      {
        Mutex::Lock l(mutex);
        return customFields.size();
      }

      Ref<CustomFieldBase> getCustomField(unsigned index) const
      {
        Mutex::Lock l(mutex);
        if(index >= customFields.size()) return 0;
        return customFields[index];
      }
      /**
       * Must only be called while holding the scenegraph read lock
       */
      Ref<CustomFieldBase> getCustomFieldNoLock(unsigned index) const
      {
        return customFields[index];
      }

      // Overriding from Node
      string dump(int level) const;

    protected:
      CustomFieldNode() {}

      vector<Ref<CustomFieldBase> > customFields;

      template<typename T> friend struct SimpleCustomField;
      template<typename T> friend struct SimpleSeqCustomField;
```

```
  template<typename C> friend struct NodeCustomField;
  template<typename C> friend struct NodeSequenceCustomField;

  /**
   * Must be called with a scenegraph write lock, and never with a
   * lock on any node. Also, this method may never be called for an
   * un-realized node.
   */
  virtual void handleEvent(int fieldIndex, Time timestamp) = 0;

private:
  // Overriding from Node
  const FieldBase *lookupField(string name) const;
};


struct CustomFieldBase: virtual RefObjectBase, virtual FieldBase
{
  int getIndex() const { return index; }

protected:
  CustomFieldBase(int index, const Node *containingNode):
    index(index)
  {
    const NodeType *t = containingNode->getType();
    unsigned long j = t->getNumberOfImmediateFields();
    j += index;
    j <<= NodeType::maxIndexBits;
    j |= t->getIndex();
    id = j;
  }

  const int index; // Index into CustomFieldNode::customFields

  // Overriding from FieldBase
  bool addRoute(Node *, const RouteTail *r) const
  {
    CustomFieldBase *f =
      const_cast<CustomFieldBase *>(this);
    return f->eventSource.addRoute(r);
  }

  // Overriding from FieldBase
  void delRoute(Node *, const RouteTail *r) const
  {
    CustomFieldBase *f =
      const_cast<CustomFieldBase *>(this);
    f->eventSource.delRoute(r);
  }

  // Overriding from FieldBase
  bool delRouteMatch(Node *, const RouteTail *r) const
  {
    CustomFieldBase *f =
      const_cast<CustomFieldBase *>(this);
    return f->eventSource.delRouteMatch(r);
  }
```



```
protected:
  EventSource eventSource;
  Time timestamp;
};

template<typename T>
struct SimpleCustomField: CustomFieldBase
{
  SimpleCustomField(const Node *containingNode,
                    int index, string name, const FieldType *type,
                    bool isEventTarget, bool isEventSource,
                    const T &v):
    FieldBase(name, type, isEventTarget, isEventSource, 0),
    CustomFieldBase(index, containingNode),
    value(v) {}

  // Overriding from FieldBase
  Ref<ValueBase> get(const Node *) const
  {
    return new SimpleValue<T>(this->getType(), value);
  }

  // Overriding from FieldBase
  void handleEvent(Node *n, const Event *e, bool cascade) const
  {
    const SimpleValueBase *val =
      dynamic_cast<const SimpleValueBase *>(e->value.get());
    if(!val) ARCHON_THROW1(InternalException,
                           "SimpleCustomField::handleEvent: "
                           "Value/Field type inconsistency");
    CustomFieldNode *m = dynamic_cast<CustomFieldNode *>(n);
    if(!m) ARCHON_THROW1(InternalException,
                         "SimpleCustomField::handleEvent: "
                         "Node was not a CustomFieldNode");
    SimpleCustomField *f =
      const_cast<SimpleCustomField *>(this);
    f->timestamp = e->time;
    f->value = *static_cast<const T *>(val->getValuePtr());
    if(cascade && isEventTarget) m->handleEvent(index, e->time);
    if(cascade) eventSource.cascadeEvent(e);
  }

  // Overriding from FieldBase
  bool changedSince(const Node *, const Time &t) const
  {
    return timestamp >= t;
  }

  // Overriding from FieldBase
  string dump(const Node *, int level) const
  {
    return string(2*level, ' ') +
      this->getName() + " " + dumpValue(value) + "\n";
  }

  void forwardClear(Node *n) const
  {
    SimpleCustomField *f =
```

```
      const_cast<SimpleCustomField *>(this);
    f->eventSource.clear();
  }

private:
  T value;
};

template<typename T>
struct SimpleSeqCustomField: CustomFieldBase, SequenceFieldBase
{
  typedef vector<T> V;

  SimpleSeqCustomField(const Node *containingNode,
                       int index, string name, const FieldType *type,
                       bool isEventTarget, bool isEventSource,
                       const V &v):
    FieldBase(name, type, isEventTarget, isEventSource, 0),
    CustomFieldBase(index, containingNode) {}

  // Overriding from FieldBase
  Ref<ValueBase> get(const Node *) const
  {
    return new SimpleValue<V>(this->getType(), value);
  }

  // Overriding from FieldBase
  void handleEvent(Node *n, const Event *e, bool cascade) const
  {
    const SimpleValueBase *val =
      dynamic_cast<const SimpleValueBase *>(e->value.get());
    if(!val) ARCHON_THROW1(InternalException,
                           "SimpleCustomField::handleEvent: "
                           "Value/Field type inconsistency");
    CustomFieldNode *m = dynamic_cast<CustomFieldNode *>(n);
    if(!m) ARCHON_THROW1(InternalException,
                         "SimpleCustomField::handleEvent: "
                         "Node was not a CustomFieldNode");
    SimpleSeqCustomField *f =
      const_cast<SimpleSeqCustomField *>(this);
    f->timestamp = e->time;
    f->value = *static_cast<const V *>(val->getValuePtr());
    if(cascade && isEventTarget) m->handleEvent(index, e->time);
    if(cascade) eventSource.cascadeEvent(e);
  }

  // Overriding from SequenceFieldBase
  unsigned getSize(const Node *) const
  {
    return value.size();
  }

  // Overriding from SequenceFieldBase
  Ref<ValueBase> getAt(const Node *, unsigned i) const
  {
    if(i >= value.size())
      ARCHON_THROW1(InternalException, "Index out of range");
```



```
    return new SimpleValue<T>(this->getType()->getSingleType(), value[i
        ]);
  }

  // Overriding from SequenceFieldBase
  void setAt(Node *n, const ValueBase *v,
             unsigned i, Time time, bool cascade) const
  {
    if(i >= value.size())
      ARCHON_THROW1(InternalException, "Index out of range");
    const SimpleValue<T> *w =
      dynamic_cast<const SimpleValue<T> *>(v);
    if(!w) ARCHON_THROW1(InternalException, "Value/Field type mismatch");
    CustomFieldNode *m = dynamic_cast<CustomFieldNode *>(n);
    if(!m) ARCHON_THROW1(InternalException, "Not a CustomFieldNode");
    SimpleSeqCustomField *f =
      const_cast<SimpleSeqCustomField *>(this);
    f->timestamp = time;
    f->value[i] = w->value;
    if(cascade && isEventTarget) m->handleEvent(index, time);
    if(cascade)
    {
      Ref<ValueBase> x = get(n);
      SimpleValue<V> *s = dynamic_cast<SimpleValue<V> *>(x.get());
      if(!s) ARCHON_THROW1(InternalException, "Wrong value type");
      Event e(s, time);
      eventSource.cascadeEvent(&e);
    }
  }

  // Overriding from SequenceFieldBase
  void add(Node *n, const ValueBase *v, Time time, bool cascade) const
  {
    const SimpleValue<T> *w =
      dynamic_cast<const SimpleValue<T> *>(v);
    if(!w) ARCHON_THROW1(InternalException, "Value/Field type mismatch");
    CustomFieldNode *m = dynamic_cast<CustomFieldNode *>(n);
    if(!m) ARCHON_THROW1(InternalException, "Not a CustomFieldNode");
    SimpleSeqCustomField *f =
      const_cast<SimpleSeqCustomField *>(this);
    f->timestamp = time;
    f->value.push_back(w->value);
    if(cascade && isEventTarget) m->handleEvent(index, time);
    if(cascade)
    {
      Ref<ValueBase> x = get(n);
      SimpleValue<V> *s = dynamic_cast<SimpleValue<V> *>(x.get());
      if(!s) ARCHON_THROW1(InternalException, "Wrong value type");
      Event e(s, time);
      eventSource.cascadeEvent(&e);
    }
  }

  // Overriding from SequenceFieldBase
  void remove(Node *n, const ValueBase *v, Time time, bool cascade) const
  {
    const SimpleValue<T> *w =
      dynamic_cast<const SimpleValue<T> *>(v);
```

```
    if(!w) ARCHON_THROW1(InternalException, "Value/Field type mismatch");
    CustomFieldNode *m = dynamic_cast<CustomFieldNode *>(n);
    if(!m) ARCHON_THROW1(InternalException, "Not a CustomFieldNode");
    SimpleSeqCustomField *f =
      const_cast<SimpleSeqCustomField *>(this);
    f->timestamp = time;
    std::remove(f->value.begin(), f->value.end(), w->value);
    if(cascade && isEventTarget) m->handleEvent(index, time);
    if(cascade)
    {
      Ref<ValueBase> x = get(n);
      SimpleValue<V> *s = dynamic_cast<SimpleValue<V> *>(x.get());
      if(!s) ARCHON_THROW1(InternalException, "Wrong value type");
      Event e(s, time);
      eventSource.cascadeEvent(&e);
    }
  }

  // Overriding from FieldBase
  bool changedSince(const Node *, const Time &t) const
  {
    return timestamp >= t;
  }

  // Overriding from FieldBase
  string dump(const Node *, int level) const
  {
    return string(2*level, ' ') +
      this->getName() + " " + dumpValue(value) + "\n";
  }

  void forwardClear(Node *n) const
  {
    SimpleSeqCustomField *f =
      const_cast<SimpleSeqCustomField *>(this);
    f->eventSource.clear();
  }

private:
  V value;
};

template<typename C>
struct NodeCustomField: CustomFieldBase, NodeFieldBase
{
  NodeCustomField(const Node *containingNode,
                  int index, string name, const NodeType *nodeType,
                  bool isEventTarget, bool isEventSource,
                  Ref<C> v):
    FieldBase(name, SFNode::type, isEventTarget, isEventSource, 0),
    CustomFieldBase(index, containingNode),
    NodeFieldBase(nodeType), value(v) {}

  // Overriding from FieldBase
  Ref<ValueBase> get(const Node *) const
  {
    return new NodeValue(SFNode::type, value.get());
  }
```



```
  // Overriding from FieldBase
  void handleEvent(Node *n, const Event *e, bool cascade) const
  {
    const NodeValue *val =
      dynamic_cast<const NodeValue *>(e->value.get());
    if(!val) ARCHON_THROW1(InternalException,
                           "NodeCustomField::handleEvent: "
                           "Value/Field type inconsistency");
    C *v = dynamic_cast<C *>(val->value.get());
    if(!v && val->value.get())
      ARCHON_THROW1(InternalException,
                    "NodeCustomField::handleEvent: "
                    "Incompatible node");
    CustomFieldNode *m = dynamic_cast<CustomFieldNode *>(n);
    if(!m) ARCHON_THROW1(InternalException,
                         "NodeCustomField::handleEvent: "
                         "Node was not a CustomFieldNode");
    NodeCustomField *f =
      const_cast<NodeCustomField *>(this);
    f->timestamp = e->time;
    f->value = v;
    if(cascade && isEventTarget) m->handleEvent(index, e->time);
    if(cascade) eventSource.cascadeEvent(e);
  }

  // Overriding from NodeFieldBase
  bool inject(Node *n, Node *child, Time time, bool cascade) const
  {
    C *c = dynamic_cast<C *>(child);
    if(!c && child)
      ARCHON_THROW1(InternalException,
                    "NodeCustomField::inject: "
                    "Incompatible node");
    CustomFieldNode *m = dynamic_cast<CustomFieldNode *>(n);
    if(!m) ARCHON_THROW1(InternalException,
                         "NodeCustomField::inject: "
                         "Node was not a CustomFieldNode");
    NodeCustomField *f =
      const_cast<NodeCustomField *>(this);
    if(f->value) return false;
    f->timestamp = time;
    f->value = c;
    if(cascade && isEventTarget) m->handleEvent(index, time);
    if(cascade)
    {
      Event e(new NodeValue(SFNode::type, c), time);
      eventSource.cascadeEvent(&e);
    }
    return true;
  }

  // Overriding from FieldBase
  bool changedSince(const Node *, const Time &t) const
  {
    return timestamp >= t || value->changedSince(t);
  }
```

```
  // Overriding from FieldBase
  string dump(const Node *, int level) const
  {
    return string(2*level, ' ') + this->getName() +
      (value ? "\n" + value->dump(level+1) : string(" NULL\n"));
  }

  void forwardClear(Node *) const
  {
    NodeCustomField *f =
      const_cast<NodeCustomField *>(this);
    f->value.reset();
    f->eventSource.clear();
  }

private:
  Ref<C> value;
};

template<typename C>
struct NodeSequenceCustomField: CustomFieldBase, NodeSequenceFieldBase
{
  NodeSequenceCustomField(const Node *containingNode,
                          int index, string name, const NodeType *
                               nodeType,
                          bool isEventTarget, bool isEventSource,
                          const vector<Ref<C> > &v):
    FieldBase(name, MFNode::type, isEventTarget, isEventSource, 0),
    CustomFieldBase(index, containingNode),
    NodeSequenceFieldBase(nodeType), value(v) {}

  // Overriding from FieldBase
  void handleEvent(Node *n, const Event *e, bool cascade) const
  {
    const NodeSequenceValue *val =
      dynamic_cast<const NodeSequenceValue *>(e->value.get());
    if(!val) ARCHON_THROW1(InternalException,
                           "NodeSequenceCustomField::handleEvent: "
                           "Value/Field type inconsistency");
    vector<Ref<C> > v;
    v.resize(val->value.size());
    for(unsigned i=0; i<val->value.size(); ++i)
    {
      Node *m = val->value[i].get();
      if(!m) ARCHON_THROW1(InternalException,
                           "NodeSequenceCustomField::handleEvent: "
                           "Null node is illegal");
      v[i] = dynamic_cast<C *>(m);
      if(!v[i])
        ARCHON_THROW1(InternalException,
                      "NodeSequenceCustomField::handleEvent: "
                      "Incompatible node");
    }
    CustomFieldNode *m = dynamic_cast<CustomFieldNode *>(n);
    if(!m) ARCHON_THROW1(InternalException,
                         "SimpleCustomField::handleEvent: "
                         "Node was not a CustomFieldNode");
    NodeSequenceCustomField *f =
```



```
      const_cast<NodeSequenceCustomField *>(this);
    f->timestamp = e->time;
    f->value = v;
    if(cascade && isEventTarget) m->handleEvent(index, e->time);
    if(cascade) eventSource.cascadeEvent(e);
  }


  Ref<ValueBase> get(const Node *) const
  {
    Ref<NodeSequenceValue> v = new NodeSequenceValue(MFNode::type);
    v->value.resize(value.size());
    for(unsigned i=0; i<value.size(); ++i) v->value[i] = value[i];
    return v;
  }

  // Overriding from SequenceFieldBase
  unsigned getSize(const Node *) const
  {
    return value.size();
  }

  // Overriding from SequenceFieldBase
  Ref<ValueBase> getAt(const Node *, unsigned i) const
  {
    if(i >= value.size())
      ARCHON_THROW1(InternalException, "Index out of range");
    return new NodeValue(SFNode::type, value[i].get());
  }

  // Overriding from SequenceFieldBase
  void setAt(Node *n, const ValueBase *v,
             unsigned i, Time time, bool cascade) const
  {
    if(i >= value.size())
      ARCHON_THROW1(InternalException, "Index out of range");
    const NodeValue *w =
      dynamic_cast<const NodeValue *>(v);
    if(!w) ARCHON_THROW1(InternalException, "Value/Field type mismatch");
    C *c = dynamic_cast<C *>(w->value.get());
    if(!c) ARCHON_THROW1(InternalException, "Incompatible node type");
    CustomFieldNode *m = dynamic_cast<CustomFieldNode *>(n);
    if(!m) ARCHON_THROW1(InternalException, "Not a CustomFieldNode");

    NodeSequenceCustomField *f =
      const_cast<NodeSequenceCustomField *>(this);
    f->timestamp = time;
    f->value[i].reset(c);
    if(cascade && isEventTarget) m->handleEvent(index, time);
    if(cascade)
    {
      Ref<ValueBase> x = get(n);
      NodeSequenceValue *s =
        dynamic_cast<NodeSequenceValue *>(x.get());
      if(!s) ARCHON_THROW1(InternalException, "Wrong value type");
      Event e(s, time);
      eventSource.cascadeEvent(&e);
    }
```

```
}

// Overriding from SequenceFieldBase
void add(Node *n, const ValueBase *v, Time time, bool cascade) const
{
  const NodeValue *w =
    dynamic_cast<const NodeValue *>(v);
  if(!w) ARCHON_THROW1(InternalException, "Value/Field type mismatch");
  C *c = dynamic_cast<C *>(w->value.get());
  if(!c) ARCHON_THROW1(InternalException, "Incompatible node type");
  CustomFieldNode *m = dynamic_cast<CustomFieldNode *>(n);
  if(!m) ARCHON_THROW1(InternalException, "Not a CustomFieldNode");

  NodeSequenceCustomField *f =
    const_cast<NodeSequenceCustomField *>(this);
  f->timestamp = time;
  f->value.push_back(c);
  if(cascade && isEventTarget) m->handleEvent(index, time);
  if(cascade)
  {
    Ref<ValueBase> x = get(n);
    NodeSequenceValue *s =
      dynamic_cast<NodeSequenceValue *>(x.get());
    if(!s) ARCHON_THROW1(InternalException, "Wrong value type");
    Event e(s, time);
    eventSource.cascadeEvent(&e);
  }
}

// Overriding from SequenceFieldBase
void remove(Node *n, const ValueBase *v, Time time, bool cascade) const
{
  const NodeValue *w =
    dynamic_cast<const NodeValue *>(v);
  if(!w) ARCHON_THROW1(InternalException, "Value/Field type mismatch");
  C *c = dynamic_cast<C *>(w->value.get());
  if(!c) ARCHON_THROW1(InternalException, "Incompatible node type");
  CustomFieldNode *m = dynamic_cast<CustomFieldNode *>(n);
  if(!m) ARCHON_THROW1(InternalException, "Not a CustomFieldNode");

  NodeSequenceCustomField *f =
    const_cast<NodeSequenceCustomField *>(this);
  f->timestamp = time;
  std::remove(f->value.begin(), f->value.end(), Ref<C>(c));
  if(cascade && isEventTarget) m->handleEvent(index, time);
  if(cascade)
  {
    Ref<ValueBase> x = get(n);
    NodeSequenceValue *s =
      dynamic_cast<NodeSequenceValue *>(x.get());
    if(!s) ARCHON_THROW1(InternalException, "Wrong value type");
    Event e(s, time);
    eventSource.cascadeEvent(&e);
  }
}


// Overriding from NodeFieldBase
```



```
bool inject(Node *n, Node *child, Time time, bool cascade) const
{
  C *c = dynamic_cast<C *>(child);
  if(!c && child)
    ARCHON_THROW1(InternalException,
                  "NodeCustomField::inject: "
                  "Incompatible node");
  CustomFieldNode *m = dynamic_cast<CustomFieldNode *>(n);
  if(!m) ARCHON_THROW1(InternalException, "Not a CustomFieldNode");
  NodeSequenceCustomField *f =
    const_cast<NodeSequenceCustomField *>(this);
  f->timestamp = time;
  f->value.push_back(c);
  if(cascade && isEventTarget) m->handleEvent(index, time);
  if(cascade)
  {
    Ref<ValueBase> x = get(n);
    NodeSequenceValue *s =
      dynamic_cast<NodeSequenceValue *>(x.get());
    if(!s) ARCHON_THROW1(InternalException, "Wrong value type");
    Event e(s, time);
    eventSource.cascadeEvent(&e);
  }
  return true;
}

// Overriding from FieldBase
bool changedSince(const Node *, const Time &t) const
{
  if(timestamp >= t) return true;
  for(unsigned i=0; i<value.size(); ++i)
  {
    const C *c = value[i].get();
    if(c->changedSince(t)) return true;
  }
  return false;
}

// Overriding from FieldBase
string dump(const Node *, int level) const
{
  string indent(2*level, ' ');
  string s = indent + this->getName();
  if(!value.size()) return s + " []\n";
  s += "\n";
  indent + "[\n";
  for(unsigned i=0; i<value.size(); ++i)
  {
    const C *child = value[i].get();
    s += child->dump(level+1);
  }
  return s + indent + "]\n";
}

void forwardClear(Node *) const
{
  NodeSequenceCustomField *f =
    const_cast<NodeSequenceCustomField *>(this);
```

```
      f->value.clear();
      f->eventSource.clear();
    }

  private:
    vector<Ref<C> > value;
  };
  }
}

#endif // ARCHON_X3D_CUSTOM_FIELD_H
```

Listing: **archon/x3d/server/custom_field.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

/*
 * \file custom_field.C
 */

#include <archon/x3d/server/custom_field.H>

namespace Archon
{
  namespace X3D
  {
    /**
     * \todo When a custom script field is named "foo" and has access
     * type 'inputOutput' may it then be accessed with the names
     * "set_foo" and "foo_changed", or is this behaivour restricted to
     * conventional fields?
     */
    bool CustomFieldNode::addCustomField(Ref<CustomFieldBase> f)
      throw(RealizedException)
    {
      if(lookupField(f->getName())) return false;
      Mutex::Lock l(mutex);
      if(isRealizedNoLock()) ARCHON_THROW(RealizedException);
      customFields.push_back(f);
      return true;
    }
```



```
    const FieldBase *CustomFieldNode::lookupField(string name) const
    {
      {
        Mutex::Lock l(mutex);
        for(unsigned i=0; i<customFields.size(); ++i)
        {
          const FieldBase *f = customFields[i].get();
          if(f->getName() == name) return f;
        }
      }
      return getType()->lookupField(name);
    }

    string CustomFieldNode::dump(int level) const
    {
      string s = getType()->dumpNode(this, level);
      string indent(2*level, ' ');
      s += indent + "Custom fields\n";
      s += indent + "{\n";
      for(unsigned i=0; i<customFields.size(); ++i)
        s += customFields[i]->dump(this, level+1);
      s += indent + "}\n";
      return s;
    }
  }
}
```

Listing: **archon/x3d/server/dump.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_DUMP_H
#define ARCHON_X3D_DUMP_H

#include <string>

#include <archon/math/vector.H>
#include <archon/math/geometry.H>
#include <archon/util/image.H>
#include <archon/util/time.H>
```

```
#include <archon/util/text.H>

namespace Archon
{
  namespace X3D
  {
    using namespace std;
    using namespace Math;
    using namespace Utilities;

    template<typename T>
    inline string dumpValue(const vector<T> &v)
    {
      string s = "[";
      for(unsigned i=0; i<v.size(); ++i)
      {
        if(i) s+= ", ";
        s += dumpValue(v[i]);
      }
      return s + "]";
    }

    inline string dumpValue(string v)
    {
      return "\"" + v + "\"";
    }

    inline string dumpValue(bool v)
    {
      return v ? "TRUE" : "FALSE";
    }

    inline string dumpValue(int v)
    {
      return Utilities::Text::toString(v);
    }

    inline string dumpValue(double v)
    {
      return Utilities::Text::toString(v);
    }

    inline string dumpValue(Vector2 v)
    {
      return
        Utilities::Text::toString(v[0]) + " " +
        Utilities::Text::toString(v[1]);
    }

    inline string dumpValue(Vector3 v)
    {
      return
        Utilities::Text::toString(v[0]) + " " +
        Utilities::Text::toString(v[1]) + " " +
        Utilities::Text::toString(v[2]);
    }

    inline string dumpValue(Vector4 v)
```



```
    {
      return
        Utilities::Text::toString(v[0]) + " " +
        Utilities::Text::toString(v[1]) + " " +
        Utilities::Text::toString(v[2]) + " " +
        Utilities::Text::toString(v[3]);
    }

    inline string dumpValue(AxisRot3 v)
    {
      return
        Utilities::Text::toString(v.normalizedAxis[0]) + " " +
        Utilities::Text::toString(v.normalizedAxis[1]) + " " +
        Utilities::Text::toString(v.normalizedAxis[2]) + " " +
        Utilities::Text::toString(v.angle);
    }

    inline string dumpValue(const Image &v)
    {
      return "<inlined image>";
    }

    inline string dumpValue(const Time &v)
    {
      return Utilities::Text::toString(v);
    }
  }
}

#endif // ARCHON_X3D_DUMP_H
```

Listing: **archon/x3d/server/event.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_EVENT_H
#define ARCHON_X3D_EVENT_H

/**
 * \file event.H
 *
```

```
 * X3D events and routes
 */

#include <archon/x3d/server/type.H>
#include <archon/x3d/server/node.H>

namespace Archon
{
  namespace X3D
  {
    struct Event;


    /**
     * Base class for tail of any route
     */
    struct RouteTail: RefObjectBase
    {
      virtual void cascadeEvent(const Event *) const = 0;

      const BackRef<Node> sourceNode;
      const FieldBase *const sourceField;

      virtual bool match(const RouteTail *) const = 0;

    protected:
      RouteTail(Ref<Node> n, const FieldBase *f):
        sourceNode(n), sourceField(f) {}

      unsigned long getTailCountNoLock() { return getUseCountNoLock(); }
    };

    /**
     * Base class for head of standard routes
     */
    struct RouteHead: virtual RefObjectBase
    {
      const BackRef<Node> targetNode;
      const FieldBase *const targetField;

      virtual Ref<Node> getSourceNode() const = 0;
      virtual const FieldBase *getSourceField() const = 0;

    protected:
      RouteHead(Ref<Node> n, const FieldBase *f):
        targetNode(n), targetField(f) {}

      unsigned long getHeadCountNoLock() { return getUseCountNoLock(); }
    };

    struct Route: RouteTail, RouteHead
    {
      struct Exception: virtual X3D::Exception {};
      struct ArgumentException: Exception, X3D::ArgumentException
      {
        ArgumentException() {}
        ArgumentException(string l, string m): Utilities::Exception(l, m) {}
      };
```



```
      /**
       * Thrown by 'add' if the source field is not an event source or
       * if the target field is not an event target.
       */
      struct AccessException: ArgumentException
      {
        AccessException(string l): Utilities::Exception(l) {}
      };
      /**
       * Thrown by 'add' if the types of the source and target fields
       * do not conform.
       */
      struct TypeException: ArgumentException
      {
        TypeException(string l): Utilities::Exception(l) {}
      };
      /**
       * Thrown by 'add' or 'del' if the source and target nodes do
       * not belong to the same context.
       */
      struct ContextException: ArgumentException
      {
        ContextException(string l): Utilities::Exception(l) {}
      };

      void cascadeEvent(const Event *e) const;

      virtual Ref<Node> getSourceNode() const;
      virtual const FieldBase *getSourceField() const;

      static void add(Ref<Node> sourceNode, const FieldBase *sourceField,
                      Ref<Node> targetNode, const FieldBase *targetField)
        throw(AccessException, TypeException, ContextException);

      static bool del(Ref<Node> sourceNode, const FieldBase *sourceField,
                      Ref<Node> targetNode, const FieldBase *targetField)
        throw(ContextException);

      void refDispose(Mutex::Lock &l);

      ~Route();

    private:
      Route(Ref<Node> sourceNode, const FieldBase *sourceField,
            Ref<Node> targetNode, const FieldBase *targetField);

      bool match(const RouteTail *) const;
    };



    struct ValueBase: virtual RefObjectBase
    {
      const FieldType *getFieldType() const { return fieldType; }

    protected:
      ValueBase(const FieldType *fieldType): fieldType(fieldType) {}
```

```
private:
  const FieldType *fieldType;
};

struct SimpleValueBase: ValueBase
{
  /**
   * \todo Get rid of it - should be really easy!
   */
  virtual const void *getValuePtr() const = 0;

protected:
  SimpleValueBase(const FieldType *fieldType): ValueBase(fieldType) {}
};

template<typename T>
struct SimpleValue: public SimpleValueBase
{
  SimpleValue(const FieldType *fieldType, const T &value):
    SimpleValueBase(fieldType), value(value) {}

  SimpleValue(const FieldType *fieldType):
    SimpleValueBase(fieldType) {}

  const void *getValuePtr() const { return &value; }

  T value;
};

struct NodeValue: public ValueBase
{
  NodeValue(const FieldType *fieldType, Node *value):
    ValueBase(fieldType), value(value) {}
  NodeValue(const FieldType *fieldType, Ref<Node> value):
    ValueBase(fieldType), value(value) {}

  Ref<Node> value;
};

struct NodeSequenceValue: public ValueBase
{
  NodeSequenceValue(const FieldType *fieldType):
    ValueBase(fieldType) {}

  vector<Ref<Node> > value;
};

struct Event
{
  Event(Ref<const ValueBase> value, const Time &time): value(value), time
       (time) {}

  const Ref<const ValueBase> value;
  const Time time;
};
```



```
  struct EventSource
  {
    /**
     * Add a route unless it is redundant.
     *
     * \return true if and only if the route was inserted.
     *
     * \note self reference is not checked for
     */
    bool addRoute(const RouteTail *);

    /**
     * Remove the specified route if it is found.
     */
    void delRoute(const RouteTail *);

    /**
     * Remove the route that matches the specified route if such a
     * route is found.
     *
     * \return true if and only if a matching route was found.
     *
     * \note The argument will normally be a proforma route created
     * with the sole intent of being compared against the real
     * routes. This is a trick to allow polymorphism among the
     * specified argument routes.
     */
    bool delRouteMatch(const RouteTail *);

    /**
     * Must be called with a write lock on the scenegraph
     */
    void cascadeEvent(const Event *e) const
    {
      vector<Ref<const RouteTail> > r(routes.size());
      vector<Ref<const RouteTail> >::iterator i=r.begin();
      {
        Mutex::Lock l(mutex);
        list<Ref<const RouteTail> >::const_iterator j=routes.begin();
        while(j!=routes.end()) *i++ = *j++;
      }
      i=r.begin();
      while(i!=r.end()) (*i++)->cascadeEvent(e);
    }

    void clear()
    {
      list<Ref<const RouteTail> > r;
      Mutex::Lock l(mutex);
      swap(routes, r);
    }

  private:
    Mutex mutex;
    list<Ref<const RouteTail> > routes;
  };
}
```

```
}

#endif // ARCHON_X3D_EVENT_H
```

Listing: archon/x3d/server/event.C

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

/**
 * \file event.C
 */

#include <archon/x3d/server/event.H>
#include <archon/x3d/server/scene.H>

namespace Archon
{
  namespace X3D
  {
    void Route::cascadeEvent(const Event *e) const
    {
      try
      {
        Ref<Node> n = targetNode.getRef();
        targetField->handleEvent(n.get(), e, true);
      }
      catch(ForwardDestroyedException &) {}
    };

    Ref<Node> Route::getSourceNode() const
    {
      return sourceNode.getRef();
    }

    const FieldBase *Route::getSourceField() const
    {
      return sourceField;
    }

    /**
     * \todo The below injective node type restriction is not
```



```
     * mentioned in the X3D specification and is probably violating
     * it.
     */
    void Route::add(Ref<Node> sourceNode, const FieldBase *sourceField,
                    Ref<Node> targetNode, const FieldBase *targetField)
      throw(AccessException, TypeException, ContextException)
    {
      // Field input/output capabilities
      if(!sourceField->getIsEventSource() ||
         !targetField->getIsEventTarget()) ARCHON_THROW(AccessException);

      // Self reference is redundant and must be ignored
      if(sourceNode == targetNode && sourceField == targetField) return;

      // Type conformance
      const FieldType *fieldType = sourceField->getType();
      if(fieldType != targetField->getType()) ARCHON_THROW(TypeException);

      // Injective node type restriction
      if(const NodeFieldBase *sourceNodeField =
         dynamic_cast<const NodeFieldBase *>(sourceField))
      {
        const NodeFieldBase *targetNodeField =
          dynamic_cast<const NodeFieldBase *>(targetField);
        if(!sourceNodeField->getNodeType()->
           isDerivedFrom(targetNodeField->getNodeType()))
          ARCHON_THROW(TypeException);
      }

      try
      {
        Ref<ExecutionContext> context = sourceNode->context.getRef();
        if(context != targetNode->context.getRef())
          ARCHON_THROW(ContextException);

        Route *r = new Route(sourceNode, sourceField, targetNode, targetField
               );
        Ref<RouteTail> t(dynamic_cast<RouteTail *>(r));
        Ref<RouteHead> h(dynamic_cast<RouteHead *>(r));
        if(sourceField->addRoute(sourceNode.get(), t.get()))
        {
          targetNode->addRouteTarget(h);
          context->add(r);
        }
      }
      catch(ForwardDestroyedException &) {}
    }

    bool Route::del(Ref<Node> sourceNode, const FieldBase *sourceField,
                    Ref<Node> targetNode, const FieldBase *targetField)
      throw(ContextException)
    {
      if(sourceNode->context != targetNode->context)
        ARCHON_THROW(ContextException);
      Route r(sourceNode, sourceField, targetNode, targetField);
      return sourceField->delRouteMatch(sourceNode.get(), &r);
    }
```

```
void Route::refDispose(Mutex::Lock &l)
{
  if(getTailCountNoLock() == 0)
  {
    if(getHeadCountNoLock() == 0)
    {
      l.release();
      try
      {
        sourceNode->context->remove(this);
      }
      catch(ForwardDestroyedException &) {}
      delete this;
      return;
    }
    Ref<RouteHead> h(this, RefNoLockTag());
    l.release();
    try
    {
      targetNode->removeRouteTarget(h);
    }
    catch(ForwardDestroyedException &) {}
    return;
  }
  Ref<RouteTail> t(this, RefNoLockTag());
  l.release();
  try
  {
    Ref<Node> n = sourceNode.getRef();
    sourceField->delRoute(n.get(), t.get());
  }
  catch(ForwardDestroyedException &) {}
}

Route::~Route() {}

Route::Route(Ref<Node> sourceNode, const FieldBase *sourceField,
             Ref<Node> targetNode, const FieldBase *targetField):
  RouteTail(sourceNode, sourceField),
  RouteHead(targetNode, targetField)
{
}

bool Route::match(const RouteTail *r) const
{
  const Route *s = dynamic_cast<const Route *>(r);
  return s && s->targetNode == targetNode &&
    s->targetField == targetField;
}


bool EventSource::addRoute(const RouteTail *r)
{
  Mutex::Lock l(mutex);
  list<Ref<const RouteTail> >::iterator i = routes.begin();
  while(i != routes.end())
  {
    if((*i)->match(r)) return false;
```



```
      ++i;
    }
    routes.push_back(r);
    return true;
  }

  void EventSource::delRoute(const RouteTail *r)
  {
    Mutex::Lock l(mutex);
    routes.remove(Ref<const RouteTail>(r));
  }

  bool EventSource::delRouteMatch(const RouteTail *r)
  {
    Mutex::Lock l(mutex);
    list<Ref<const RouteTail> >::iterator i = routes.begin();
    while(i != routes.end())
    {
      if((*i)->match(r))
      {
        routes.erase(i);
        return true;
      }
      ++i;
    }
    return false;
  }
 }
}
```

Listing: **archon/x3d/server/exception.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_EXCEPTION_H
#define ARCHON_X3D_EXCEPTION_H

#include <string>

#include <archon/util/exception.H>
```

```
namespace Archon
{
  namespace X3D
  {
    using namespace std;

    struct Exception: virtual Utilities::Exception {};

    /**
     * A problem with one or more of the method arguments
     */
    struct ArgumentException: virtual Exception, Utilities::ArgumentException
    {
      ArgumentException() {}
      ArgumentException(string l): Utilities::Exception(l) {}
      ArgumentException(string l, string m): Utilities::Exception(l, m) {}
    };

    /**
     * A problem with getting access to external resources
     */
    struct ResourceException: virtual Exception, Utilities::ResourceException
    {
      ResourceException() {}
      ResourceException(string l): Utilities::Exception(l) {}
      ResourceException(string l, string m): Utilities::Exception(l, m) {}
    };

    /**
     * The request operation is not compatible with the current state
     * of the object.
     */
    struct StateException: virtual Exception, Utilities::StateException
    {
      StateException() {}
      StateException(string l): Utilities::Exception(l) {}
      StateException(string l, string m): Utilities::Exception(l, m) {}
    };

    /**
     * Internal error encountered
     */
    struct InternalException: virtual Exception, Utilities::InternalException
    {
      InternalException() {}
      InternalException(string l): Utilities::Exception(l) {}
      InternalException(string l, string m): Utilities::Exception(l, m) {}
    };
  }
}

#endif // ARCHON_X3D_EXCEPTION_H
```

Listing: **archon/x3d/server/field.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
```



```
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_FIELD_H
#define ARCHON_X3D_FIELD_H

/**
 * \file field.H
 *
 * Specific X3D field types.
 */

#include <archon/x3d/server/field_type.H>
#include <archon/x3d/server/type.H>
#include <archon/x3d/server/event.H>
#include <archon/x3d/server/dump.H>

namespace Archon
{
  namespace X3D
  {
    template<class N, typename T>
    struct StaticField: virtual FieldBase
    {
    protected:
      StaticField(T N::*var, void (N::*func)(const T &),
                  EventSource N::*eventSource, Time N::*stamp):
        var(var), func(func), eventSource(eventSource), stamp(stamp)
      {
        if(!var && (getIsEventSource() || !getIsEventTarget()) ||
           func && getIsEventSource() && !getIsEventTarget() ||
           bool(eventSource) != getIsEventSource() ||
           bool(stamp) != getIsEventTarget())
          ARCHON_THROW1(ArgumentException, "Bad StaticField '" + getName() +
               "'");
      }

      bool addRoute(Node *n, const RouteTail *r) const
      {
        return (dynamic_cast<N *>(n)->*eventSource).addRoute(r);
      }

      void delRoute(Node *n, const RouteTail *r) const
      {
        (dynamic_cast<N *>(n)->*eventSource).delRoute(r);
```

```
  }

  bool delRouteMatch(Node *n, const RouteTail *r) const
  {
    return (dynamic_cast<N *>(n)->*eventSource).delRouteMatch(r);
  }

protected:
  T N::*var;
  void (N::*func)(const T &);
  EventSource N::*eventSource;
  Time N::*stamp;
};

template<class N, typename T>
struct SimpleField: StaticField<N, T>
{
  SimpleField(string name, const FieldType *type,
              bool isEventTarget, bool isEventSource,
              T N::*var,
              void (N::*func)(const T &),
              EventSource N::*eventSource,
              Time N::*stamp,
              unsigned flags):
    FieldBase(name, type, isEventTarget, isEventSource, flags),
    StaticField<N, T>(var, func, eventSource, stamp) {}

  Ref<ValueBase> get(const Node *n) const
  {
    if(!var) ARCHON_THROW1(InternalException, "No variable");
    return new SimpleValue<T>(getType(), dynamic_cast<const N *>(n)->*var
        );
  }

  void handleEvent(Node *n, const Event *e, bool cascade) const
  {
    const SimpleValueBase *val =
      dynamic_cast<const SimpleValueBase *>(e->value.get());
    if(!val) ARCHON_THROW1(InternalException,
                           "SimpleField::handleEvent: "
                           "Value/Field type inconsistency");
    const T &v = *static_cast<const T *>(val->getValuePtr());
    if(stamp) dynamic_cast<N *>(n)->*stamp = e->time;
    if(var) dynamic_cast<N *>(n)->*var = v;
    if(func) (dynamic_cast<N *>(n)->*func)(v);
    if(cascade && eventSource)
      (dynamic_cast<N *>(n)->*eventSource).cascadeEvent(e);
  }

  /**
   * Is this field or any field of child nodes of this field changed?
   * Child nodes of course are only relevant for node fields.
   */
  bool changedSince(const Node *n, const Time &t) const
  {
    return stamp && dynamic_cast<const N *>(n)->*stamp >= t;
  }
```



```
  string dump(const Node *n, int level) const
  {
    if(!var) return "";
    return
      string(2*level, ' ') + getName() + " " +
      dumpValue(dynamic_cast<const N *>(n)->*var) + "\n";
  }

  void forwardClear(Node *n) const
  {
    if(eventSource) (dynamic_cast<N *>(n)->*eventSource).clear();
  }

protected:
  SimpleField(T N::*var,
              void (N::*func)(const T &),
              EventSource N::*eventSource,
              Time N::*stamp):
    StaticField<N, T>(var, func, eventSource, stamp) {}
};

template<class N, typename T>
struct SimpleSequenceField: SimpleField<N, vector<T> >, SequenceFieldBase
{
  typedef vector<T> V;

  SimpleSequenceField(string name, const FieldType *type,
                      bool isEventTarget, bool isEventSource,
                      V N::*var,
                      void (N::*func)(const V &),
                      EventSource N::*eventSource,
                      Time N::*stamp,
                      unsigned flags):
    FieldBase(name, type, isEventTarget, isEventSource, flags),
    SimpleField<N, V>(var, func, eventSource, stamp) {}

  unsigned getSize(const Node *n) const
  {
    if(!var) ARCHON_THROW1(InternalException, "No variable");
    return (dynamic_cast<const N *>(n)->*var).size();
  }

  Ref<ValueBase> getAt(const Node *n, unsigned i) const
  {
    if(i >= getSize(n))
      ARCHON_THROW1(InternalException, "Index out of range");
    return new SimpleValue<T>
      (getType()->getSingleType(), (dynamic_cast<const N *>(n)->*var)[i
          ]);
  }

  void setAt(Node *n, const ValueBase *v,
             unsigned i, Time time, bool cascade) const
  {
    if(i >= getSize(n))
      ARCHON_THROW1(InternalException, "Index out of range");
    const SimpleValue<T> *w = dynamic_cast<const SimpleValue<T> *>(v);
    if(!w) ARCHON_THROW1(InternalException, "Value/Field type mismatch");
```

```
    if(!var) ARCHON_THROW1(InternalException, "No variable");
    if(stamp) dynamic_cast<N *>(n)->*stamp = time;
    (dynamic_cast<N *>(n)->*var)[i] = w->value;
    if(func)
    {
      V u = dynamic_cast<N *>(n)->*var;
      (dynamic_cast<N *>(n)->*func)(u);
    }
    if(cascade && eventSource)
    {
      Ref<ValueBase> x = get(n);
      SimpleValue<V> *s =
        dynamic_cast<SimpleValue<V> *>(x.get());
      if(!s) ARCHON_THROW1(InternalException, "Wrong value type");
      Event e(s, time);
      (dynamic_cast<N *>(n)->*eventSource).cascadeEvent(&e);
    }
  }

  void add(Node *n, const ValueBase *v, Time time, bool cascade) const
  {
    const SimpleValue<T> *w = dynamic_cast<const SimpleValue<T> *>(v);
    if(!w) ARCHON_THROW1(InternalException, "Value/Field type mismatch");
    if(!var) ARCHON_THROW1(InternalException, "No variable");
    if(stamp) dynamic_cast<N *>(n)->*stamp = time;
    (dynamic_cast<N *>(n)->*var).push_back(w->value);
    if(func)
    {
      V u = dynamic_cast<N *>(n)->*var;
      (dynamic_cast<N *>(n)->*func)(u);
    }
    if(cascade && eventSource)
    {
      Ref<ValueBase> x = get(n);
      SimpleValue<V> *s =
        dynamic_cast<SimpleValue<V> *>(x.get());
      if(!s) ARCHON_THROW1(InternalException, "Wrong value type");
      Event e(s, time);
      (dynamic_cast<N *>(n)->*eventSource).cascadeEvent(&e);
    }
  }

  void remove(Node *n, const ValueBase *v, Time time, bool cascade) const
  {
    const SimpleValue<T> *w = dynamic_cast<const SimpleValue<T> *>(v);
    if(!w) ARCHON_THROW1(InternalException, "Value/Field type mismatch");
    if(!var) ARCHON_THROW1(InternalException, "No variable");
    if(stamp) dynamic_cast<N *>(n)->*stamp = time;
    {
      V &u = dynamic_cast<N *>(n)->*var;
      u.erase(std::remove(u.begin(), u.end(), w->value), u.end());
    }
    if(func)
    {
      V u = dynamic_cast<N *>(n)->*var;
      (dynamic_cast<N *>(n)->*func)(u);
    }
    if(cascade && eventSource)
```



```
    {
      Ref<ValueBase> x = get(n);
      SimpleValue<V> *s =
        dynamic_cast<SimpleValue<V> *>(x.get());
      if(!s) ARCHON_THROW1(InternalException, "Wrong value type");
      Event e(s, time);
      (dynamic_cast<N *>(n)->*eventSource).cascadeEvent(&e);
    }
  }
};

template<class N, typename C>
struct NodeField: NodeFieldBase, StaticField<N, Ref<C> >
{
  NodeField(string name,
            const NodeType *nodeType,
            bool isEventTarget, bool isEventSource,
            Ref<C> N::*var,
            void (N::*func)(const Ref<C> &),
            EventSource N::*eventSource,
            Time N::*stamp,
            unsigned flags):
    FieldBase(name, SFNode::type, isEventTarget, isEventSource, flags),
    NodeFieldBase(nodeType),
    StaticField<N, Ref<C> >(var, func, eventSource, stamp) {}

  Ref<ValueBase> get(const Node *n) const
  {
    if(!var) ARCHON_THROW1(InternalException, "No variable");
    return new NodeValue(SFNode::type,
                         (dynamic_cast<const N *>(n)->*var).get());
  }

  void handleEvent(Node *n, const Event *e, bool cascade) const
  {
    const NodeValue *val =
      dynamic_cast<const NodeValue *>(e->value.get());
    if(!val) ARCHON_THROW1(InternalException,
                           "NodeField::handleEvent: "
                           "Value/Field type inconsistency");
    C *v = dynamic_cast<C *>(val->value.get());
    if(!v && val->value.get())
      ARCHON_THROW1(InternalException,
                    "NodeField::handleEvent: "
                    "Incompatible node");
    if(stamp) dynamic_cast<N *>(n)->*stamp = e->time;
    if(var) dynamic_cast<N *>(n)->*var = v;
    if(func) (dynamic_cast<N *>(n)->*func)(v);
    if(cascade && eventSource)
      (dynamic_cast<N *>(n)->*eventSource).cascadeEvent(e);
  }

  bool inject(Node *n, Node *child, Time time, bool cascade) const
  {
    C *c = dynamic_cast<C *>(child);
    if(!c && child)
      ARCHON_THROW1(InternalException,
                    "NodeField::inject: "
```

```
                    "Incompatible node");
    if(!var) return false;
    if(dynamic_cast<N *>(n)->*var) return false;
    if(stamp) dynamic_cast<N *>(n)->*stamp = time;
    if(var) dynamic_cast<N *>(n)->*var = c;
    if(func) (dynamic_cast<N *>(n)->*func)(c);
    if(cascade && eventSource)
    {
      Event e(new NodeValue(SFNode::type, c), time);
      (dynamic_cast<N *>(n)->*eventSource).cascadeEvent(&e);
    }
    return true;
  }

  /**
   * Is this field or any field of child nodes of this field changed?
   * Child nodes of course are only relevant for node fields.
   */
  bool changedSince(const Node *n, const Time &t) const
  {
    const N *m = dynamic_cast<const N *>(n);
    if(stamp && m->*stamp >= t) return true;
    if(!var) return false;
    const C *c = (m->*var).get();
    return c && c->changedSince(t);
  }

  string dump(const Node *n, int level) const
  {
    if(!var) return "";
    Node *child = (dynamic_cast<const N *>(n)->*var).get();
    return
      string(2*level, ' ') + getName() +
      (child ? "\n" + child->dump(level+1) : string(" NULL\n"));
  }

  void forwardClear(Node *n) const
  {
    if(var) (dynamic_cast<N *>(n)->*var).reset();
    if(eventSource) (dynamic_cast<N *>(n)->*eventSource).clear();
  }
};


template<class N, typename C>
struct NodeSequenceField: NodeSequenceFieldBase, StaticField<N, vector<
     Ref<C> > >
{
  NodeSequenceField(string name,
                    const NodeType *nodeType,
                    bool isEventTarget, bool isEventSource,
                    vector<Ref<C> > N::*var,
                    void (N::*func)(const vector<Ref<C> > &),
                    EventSource N::*eventSource,
                    Time N::*stamp,
                    unsigned flags):
    FieldBase(name, MFNode::type, isEventTarget, isEventSource, flags),
    NodeSequenceFieldBase(nodeType),
```



```
    StaticField<N, vector<Ref<C> > >(var, func, eventSource, stamp) {}

  void handleEvent(Node *n, const Event *e, bool cascade) const
  {
    const NodeSequenceValue *val =
      dynamic_cast<const NodeSequenceValue *>(e->value.get());
    if(!val) ARCHON_THROW1(InternalException,
                           "NodeSequenceField::handleEvent: "
                           "Value/Field type inconsistency");
    vector<Ref<C> > v;
    v.resize(val->value.size());
    for(unsigned i=0; i<val->value.size(); ++i)
    {
      Node *m = val->value[i].get();
      if(!m) ARCHON_THROW1(InternalException,
                           "NodeSequenceField::handleEvent: "
                           "Null node is illegal");
      v[i] = dynamic_cast<C *>(m);
      if(!v[i])
        ARCHON_THROW1(InternalException,
                      "NodeSequenceField::handleEvent: "
                      "Incompatible node");
    }
    if(stamp) dynamic_cast<N *>(n)->*stamp = e->time;
    if(var) dynamic_cast<N *>(n)->*var = v;
    if(func) (dynamic_cast<N *>(n)->*func)(v);
    if(cascade && eventSource)
      (dynamic_cast<N *>(n)->*eventSource).cascadeEvent(e);
  }

  Ref<ValueBase> get(const Node *n) const
  {
    if(!var) ARCHON_THROW1(InternalException, "No variable");
    Ref<NodeSequenceValue> v = new NodeSequenceValue(MFNode::type);
    const vector<Ref<C> > &w = dynamic_cast<const N *>(n)->*var;
    v->value.resize(w.size());
    for(unsigned i=0; i<w.size(); ++i) v->value[i] = w[i];
    return v;
  }

  unsigned getSize(const Node *n) const
  {
    if(!var) ARCHON_THROW1(InternalException, "No variable");
    return (dynamic_cast<const N *>(n)->*var).size();
  }

  Ref<ValueBase> getAt(const Node *n, unsigned i) const
  {
    if(i >= getSize(n))
      ARCHON_THROW1(InternalException, "Index out of range");
    return new NodeValue(SFNode::type,
                         (dynamic_cast<const N *>(n)->*var)[i].get());
  }

  void setAt(Node *n, const ValueBase *v,
             unsigned i, Time time, bool cascade) const
  {
    if(i >= getSize(n))
```

```
      ARCHON_THROW1(InternalException, "Index out of range");
    const NodeValue *w = dynamic_cast<const NodeValue *>(v);
    if(!w) ARCHON_THROW1(InternalException, "Value/Field type mismatch");
    C *c = dynamic_cast<C *>(w->value.get());
    if(!c) ARCHON_THROW1(InternalException, "Incompatible node type");
    if(!var) ARCHON_THROW1(InternalException, "No variable");

    if(stamp) dynamic_cast<N *>(n)->*stamp = time;
    (dynamic_cast<N *>(n)->*var)[i].reset(c);
    if(func)
    {
      vector<Ref<C> > u = dynamic_cast<N *>(n)->*var;
      (dynamic_cast<N *>(n)->*func)(u);
    }
    if(cascade && eventSource)
    {
      Ref<ValueBase> x = get(n);
      NodeSequenceValue *s =
        dynamic_cast<NodeSequenceValue *>(x.get());
      if(!s) ARCHON_THROW1(InternalException, "Wrong value type");
      Event e(s, time);
      (dynamic_cast<N *>(n)->*eventSource).cascadeEvent(&e);
    }
  }

  void add(Node *n, const ValueBase *v, Time time, bool cascade) const
  {
    const NodeValue *w = dynamic_cast<const NodeValue *>(v);
    if(!w) ARCHON_THROW1(InternalException, "Value/Field type mismatch");
    C *c = dynamic_cast<C *>(w->value.get());
    if(!c) ARCHON_THROW1(InternalException, "Incompatible node type");
    if(!var) ARCHON_THROW1(InternalException, "No variable");

    if(stamp) dynamic_cast<N *>(n)->*stamp = time;
    (dynamic_cast<N *>(n)->*var).push_back(c);
    if(func)
    {
      vector<Ref<C> > u = dynamic_cast<N *>(n)->*var;
      (dynamic_cast<N *>(n)->*func)(u);
    }
    if(cascade && eventSource)
    {
      Ref<ValueBase> x = get(n);
      NodeSequenceValue *s =
        dynamic_cast<NodeSequenceValue *>(x.get());
      if(!s) ARCHON_THROW1(InternalException, "Wrong value type");
      Event e(s, time);
      (dynamic_cast<N *>(n)->*eventSource).cascadeEvent(&e);
    }
  }

  void remove(Node *n, const ValueBase *v, Time time, bool cascade) const
  {
    const NodeValue *w = dynamic_cast<const NodeValue *>(v);
    if(!w) ARCHON_THROW1(InternalException, "Value/Field type mismatch");
    C *c = dynamic_cast<C *>(w->value.get());
    if(!c) ARCHON_THROW1(InternalException, "Incompatible node type");
    if(!var) ARCHON_THROW1(InternalException, "No variable");
```



```
    if(stamp) dynamic_cast<N *>(n)->*stamp = time;
    {
      vector<Ref<C> > &u = dynamic_cast<N *>(n)->*var;
      u.erase(std::remove(u.begin(), u.end(), Ref<C>(c)), u.end());
    }
    if(func)
    {
      vector<Ref<C> > u = dynamic_cast<N *>(n)->*var;
      (dynamic_cast<N *>(n)->*func)(u);
    }
    if(cascade && eventSource)
    {
      Ref<ValueBase> x = get(n);
      NodeSequenceValue *s =
        dynamic_cast<NodeSequenceValue *>(x.get());
      if(!s) ARCHON_THROW1(InternalException, "Wrong value type");
      Event e(s, time);
      (dynamic_cast<N *>(n)->*eventSource).cascadeEvent(&e);
    }
  }

  bool inject(Node *n, Node *child, Time time, bool cascade) const
  {
    C *c = dynamic_cast<C *>(child);
    if(!c && child)
      ARCHON_THROW1(InternalException,
                    "NodeField::inject: "
                    "Incompatible node");
    if(!var) return false;
    if(stamp) dynamic_cast<N *>(n)->*stamp = time;
    (dynamic_cast<N *>(n)->*var).push_back(c);
    if(func)
    {
      vector<Ref<C> > u = dynamic_cast<N *>(n)->*var;
      (dynamic_cast<N *>(n)->*func)(u);
    }
    if(cascade && eventSource)
    {
      Ref<ValueBase> x = get(n);
      NodeSequenceValue *s =
        dynamic_cast<NodeSequenceValue *>(x.get());
      if(!s) ARCHON_THROW1(InternalException, "Wrong value type");
      Event e(s, time);
      (dynamic_cast<N *>(n)->*eventSource).cascadeEvent(&e);
    }
    return true;
  }

  /**
   * Is this field or any field of child nodes of this field changed?
   * Child nodes of course are only relevant for node fields.
   */
  bool changedSince(const Node *n, const Time &t) const
  {
    const N *m = dynamic_cast<const N *>(n);
    if(stamp && m->*stamp >= t) return true;
    if(!var) return false;
```

```
    const vector<Ref<C> > &v = m->*var;
    for(unsigned i=0; i<v.size(); ++i)
    {
      const C *c = v[i].get();
      if(c && c->changedSince(t)) return true;
    }
    return false;
  }

  string dump(const Node *n, int level) const
  {
    if(!var) return "";
    const vector<Ref<C> > &v = dynamic_cast<const N *>(n)->*var;
    string indent(2*level, ' ');
    string s = indent + getName();
    if(!v.size()) return s + " []\n";
    s += "\n";
    indent + "[\n";
    for(unsigned i=0; i<v.size(); ++i)
    {
      const C *child = v[i].get();
      s += child->dump(level+1);
    }
    return s + indent + "]\n";
  }

  void forwardClear(Node *n) const
  {
    if(var) (dynamic_cast<N *>(n)->*var).clear();
    if(eventSource) (dynamic_cast<N *>(n)->*eventSource).clear();
  }
};



// SimpleField

template<class N, typename T>
const SimpleField<N, T> *newPrivateField(string n, const FieldType *t,
                                         T N::*v)
{
  return new SimpleField<N, T>(n, t, false, false, v, 0, 0, 0, 0);
}

template<class N, typename T>
const SimpleField<N, T> *newEventIn(string n, const FieldType *t, T N::*v
     ,
                                    Time N::*p, unsigned flags = 0)
{
  return new SimpleField<N, T>(n, t, true, false, v, 0, 0, p, flags);
}

template<class N, typename T>
const SimpleField<N, T> *newEventIn(string n, const FieldType *t,
                                    void (N::*f)(const T &),
                                    Time N::*p, unsigned flags = 0)
{
  return new SimpleField<N, T>(n, t, true, false, 0, f, 0, p, flags);
```



```
}

template<class N, typename T>
const SimpleField<N, T> *newExposedField(string n, const FieldType *t,
                                         T N::*v, EventSource N::*s,
                                         Time N::*p)
{
  return new SimpleField<N, T>(n, t, true, true, v, 0, s, p, 0);
}

template<class N, typename T>
const SimpleField<N, T> *newExposedField(string n, const FieldType *t,
                                         T N::*v, void (N::*f)(const T
                                                &),
                                         EventSource N::*s, Time N::*p)
{
  return new SimpleField<N, T>(n, t, true, true, v, f, s, p, 0);
}

template<class N, typename T>
const SimpleField<N, T> *newEventOut(string n, const FieldType *t,
                                     T N::*v, EventSource N::*s)
{
  return new SimpleField<N, T>(n, t, false, true, v, 0, s, 0, 0);
}



// SimpleSequenceField

template<class N, typename T>
const SimpleSequenceField<N, T> *
newPrivateField(string n, const FieldType *t, vector<T> N::*v)
{
  return new SimpleSequenceField<N, T>(n, t, false, false,
                                       v, 0, 0, 0, 0);
}

template<class N, typename T>
const SimpleSequenceField<N, T> *
newEventIn(string n, const FieldType *t, vector<T> N::*v, Time N::*p,
           unsigned flags = 0)
{
  return new SimpleSequenceField<N, T>(n, t, true, false,
                                       v, 0, 0, p, flags);
}

template<class N, typename T>
const SimpleSequenceField<N, T> *
newEventIn(string n, const FieldType *t, void (N::*f)(const vector<T> &),
           Time N::*p, unsigned flags = 0)
{
  return new SimpleSequenceField<N, T>(n, t, true, false,
                                       0, f, 0, p, flags);
}

template<class N, typename T>
const SimpleSequenceField<N, T> *
```

```
newExposedField(string n, const FieldType *t, vector<T> N::*v,
                EventSource N::*s, Time N::*p)
{
  return new SimpleSequenceField<N, T>(n, t, true, true,
                                       v, 0, s, p, 0);
}

template<class N, typename T>
const SimpleSequenceField<N, T> *
newExposedField(string n, const FieldType *t, vector<T> N::*v,
                void (N::*f)(const vector<T> &), EventSource N::*s,
                Time N::*p)
{
  return new SimpleSequenceField<N, T>(n, t, true, true,
                                       v, f, s, p, 0);
}

template<class N, typename T>
const SimpleSequenceField<N, T> *
newEventOut(string n, const FieldType *t, vector<T> N::*v,
            EventSource N::*s)
{
  return new SimpleSequenceField<N, T>(n, t, false, true,
                                       v, 0, s, 0, 0);
}



// NodeField

template<class N, typename C>
const NodeField<N, C> *newPrivateField(string n, Ref<C> N::*v)
{
  return new NodeField<N, C>(n, C::type, false, false, v, 0, 0, 0, 0);
}

template<class N, typename C>
const NodeField<N, C> *newEventIn(string n, Ref<C> N::*v, Time N::*p,
                                  unsigned flags = 0)
{
  return new NodeField<N, C>(n, C::type, true, false, v, 0, 0, p, flags);
}

template<class N, typename C>
const NodeField<N, C> *newEventIn(string n, void (N::*f)(const Ref<C> &),
                                  Time N::*p, unsigned flags = 0)
{
  return new NodeField<N, C>(n, C::type, true, false, 0, f, 0, p, flags);
}

template<class N, typename C>
const NodeField<N, C> *newExposedField(string n, Ref<C> N::*v,
                                       EventSource N::*s, Time N::*p)
{
  return new NodeField<N, C>(n, C::type, true, true, v, 0, s, p, 0);
}

template<class N, typename C>
```



```
const NodeField<N, C> *newExposedField(string n, Ref<C> N::*v,
                                       void (N::*f)(const Ref<C> &),
                                       EventSource N::*s, Time N::*p)
{
  return new NodeField<N, C>(n, C::type, true, true, v, f, s, p, 0);
}

template<class N, typename C>
const NodeField<N, C> *newEventOut(string n, Ref<C> N::*v,
                                   EventSource N::*s)
{
  return new NodeField<N, C>(n, C::type, false, true, v, 0, s, 0, 0);
}



// NodeSequenceField

template<class N, typename C>
const NodeSequenceField<N, C> *
newPrivateField(string n, vector<Ref<C> > N::*v)
{
  return new NodeSequenceField<N, C>(n, C::type, false, false,
                                     v, 0, 0, 0, 0);
}

template<class N, typename C>
const NodeSequenceField<N, C> *
newEventIn(string n, vector<Ref<C> > N::*v, Time N::*p,
           unsigned flags = 0)
{
  return new NodeSequenceField<N, C>(n, C::type, true, false,
                                     v, 0, 0, p, flags);
}

template<class N, typename C>
const NodeSequenceField<N, C> *
newEventIn(string n, void (N::*f)(const vector<Ref<C> > &), Time N::*p,
           unsigned flags = 0)
{
  return new NodeSequenceField<N, C>(n, C::type, true, false,
                                     0, f, 0, p, flags);
}

template<class N, typename C>
const NodeSequenceField<N, C> *
newExposedField(string n, vector<Ref<C> > N::*v, EventSource N::*s,
                Time N::*p)
{
  return new NodeSequenceField<N, C>(n, C::type, true, true,
                                     v, 0, s, p, 0);
}

template<class N, typename C>
const NodeSequenceField<N, C> *
newExposedField(string n, vector<Ref<C> > N::*v,
                void (N::*f)(const vector<Ref<C> > &),
                EventSource N::*s, Time N::*p)
```

```
    {
      return new NodeSequenceField<N, C>(n, C::type, true, true,
                                         v, f, s, p, 0);
    }

    template<class N, typename C>
    const NodeSequenceField<N, C> *
    newEventOut(string n, vector<Ref<C> > N::*v, EventSource N::*s)
    {
      return new NodeSequenceField<N, C>(n, C::type, false, true,
                                         v, 0, s, 0, 0);
    }
  }
}

#endif // ARCHON_X3D_FIELD_H
```

Listing: **archon/x3d/server/field_type.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_FIELD_TYPE_H
#define ARCHON_X3D_FIELD_TYPE_H

/**
 * \file field_type.H
 *
 * X3D field types
 */

#include <string>
#include <vector>

#include <archon/math/vector.H>
#include <archon/math/geometry.H>

#include <archon/util/ref.H>
#include <archon/util/time.H>
#include <archon/util/image.H>

#include <archon/x3d/server/exception.H>
```



```
#include <archon/x3d/server/event.H>

namespace Archon
{
  namespace X3D
  {
    using namespace std;
    using namespace Math;
    using namespace Utilities;

    struct SFBool
    {
      typedef bool Type;
      typedef Type ArgType;
      typedef SimpleValue<Type> Value;
      static const unsigned index = 0;
      static const FieldType *type;
    };

    struct MFBool
    {
      typedef SFBool Single;
      typedef vector<Single::Type> Type;
      typedef const Type &ArgType;
      typedef SimpleValue<Type> Value;
      static const unsigned index = 1;
      static const FieldType *type;
    };


    struct SFColor
    {
      typedef double ElemType;
      static const unsigned size = 3;
      typedef BasicVector<ElemType, size> Type;
      typedef Type ArgType;
      typedef SimpleValue<Type> Value;
      static const unsigned index = 2;
      static const FieldType *type;
    };

    struct MFColor
    {
      typedef SFColor Single;
      typedef vector<Single::Type> Type;
      typedef const Type &ArgType;
      typedef SimpleValue<Type> Value;
      static const unsigned index = 3;
      static const FieldType *type;
    };


    struct SFColorRGBA
    {
      typedef double ElemType;
      static const unsigned size = 4;
      typedef BasicVector<ElemType, size> Type;
      typedef Type ArgType;
```

```
  typedef SimpleValue<Type> Value;
  static const unsigned index = 4;
  static const FieldType *type;
};

struct MFColorRGBA
{
  typedef SFColorRGBA Single;
  typedef vector<Single::Type> Type;
  typedef const Type &ArgType;
  typedef SimpleValue<Type> Value;
  static const unsigned index = 5;
  static const FieldType *type;
};


struct SFDouble
{
  typedef double Type;
  typedef Type ArgType;
  typedef SimpleValue<Type> Value;
  static const unsigned index = 6;
  static const FieldType *type;
};

struct MFDouble
{
  typedef SFDouble Single;
  typedef vector<Single::Type> Type;
  typedef const Type &ArgType;
  typedef SimpleValue<Type> Value;
  static const unsigned index = 7;
  static const FieldType *type;
};


struct SFFloat
{
  typedef double Type;
  typedef Type ArgType;
  typedef SimpleValue<Type> Value;
  static const unsigned index = 8;
  static const FieldType *type;
};

struct MFFloat
{
  typedef SFFloat Single;
  typedef vector<Single::Type> Type;
  typedef const Type &ArgType;
  typedef SimpleValue<Type> Value;
  static const unsigned index = 9;
  static const FieldType *type;
};


struct SFImage
{
```



```
  typedef Image Type;
  typedef const Type &ArgType;
  typedef SimpleValue<Type> Value;
  static const unsigned index = 10;
  static const FieldType *type;
};

struct MFImage
{
  typedef SFImage Single;
  typedef vector<Single::Type> Type;
  typedef const Type &ArgType;
  typedef SimpleValue<Type> Value;
  static const unsigned index = 11;
  static const FieldType *type;
};


struct SFInt32
{
  typedef int Type;
  typedef Type ArgType;
  typedef SimpleValue<Type> Value;
  static const unsigned index = 12;
  static const FieldType *type;
};

struct MFInt32
{
  typedef SFInt32 Single;
  typedef vector<Single::Type> Type;
  typedef const Type &ArgType;
  typedef SimpleValue<Type> Value;
  static const unsigned index = 13;
  static const FieldType *type;
};


struct SFNode
{
  static const unsigned index = 14;
  static const FieldType *type;
};

struct MFNode
{
  static const unsigned index = 15;
  static const FieldType *type;
};

template<typename C> struct SFNodeVar: SFNode
{
  typedef Ref<C> Type;
  typedef Type ArgType;
  typedef NodeValue Value;
};

template<typename C> struct MFNodeVar: MFNode
```

```
{
  typedef SFNodeVar<C> Single;
  typedef vector<typename Single::Type> Type;
  typedef const Type &ArgType;
  typedef NodeSequenceValue Value;
};


struct SFRotation
{
  typedef double ElemType;
  static const unsigned size = 4;
  typedef AxisRot3 Type;
  typedef Type ArgType;
  typedef SimpleValue<Type> Value;
  static const unsigned index = 16;
  static const FieldType *type;
};

struct MFRotation
{
  typedef SFRotation Single;
  typedef vector<Single::Type> Type;
  typedef const Type &ArgType;
  typedef SimpleValue<Type> Value;
  static const unsigned index = 17;
  static const FieldType *type;
};


struct SFString
{
  typedef string Type;
  typedef Type ArgType;
  typedef SimpleValue<Type> Value;
  static const unsigned index = 18;
  static const FieldType *type;
};

struct MFString
{
  typedef SFString Single;
  typedef vector<Single::Type> Type;
  typedef const Type &ArgType;
  typedef SimpleValue<Type> Value;
  static const unsigned index = 19;
  static const FieldType *type;
};


struct SFTime
{
  typedef long ElemType;
  static const unsigned size = 2;
  typedef Time Type;
  typedef Type ArgType;
  typedef SimpleValue<Type> Value;
  static const unsigned index = 20;
```



```
  static const FieldType *type;
};

struct MFTime
{
  typedef SFTime Single;
  typedef vector<Single::Type> Type;
  typedef const Type &ArgType;
  typedef SimpleValue<Type> Value;
  static const unsigned index = 21;
  static const FieldType *type;
};


struct SFVec2d
{
  typedef SFDouble::Type ElemType;
  static const unsigned size = 2;
  typedef BasicVector<ElemType, size> Type;
  typedef Type ArgType;
  typedef SimpleValue<Type> Value;
  static const unsigned index = 22;
  static const FieldType *type;
};

struct MFVec2d
{
  typedef SFVec2d Single;
  typedef vector<Single::Type> Type;
  typedef const Type &ArgType;
  typedef SimpleValue<Type> Value;
  static const unsigned index = 23;
  static const FieldType *type;
};


struct SFVec2f
{
  typedef SFFloat::Type ElemType;
  static const unsigned size = 2;
  typedef BasicVector<ElemType, size> Type;
  typedef Type ArgType;
  typedef SimpleValue<Type> Value;
  static const unsigned index = 24;
  static const FieldType *type;
};

struct MFVec2f
{
  typedef SFVec2f Single;
  typedef vector<Single::Type> Type;
  typedef const Type &ArgType;
  typedef SimpleValue<Type> Value;
  static const unsigned index = 25;
  static const FieldType *type;
};
```

```
    struct SFVec3d
    {
      typedef SFDouble::Type ElemType;
      static const unsigned size = 3;
      typedef BasicVector<ElemType, size> Type;
      typedef Type ArgType;
      typedef SimpleValue<Type> Value;
      static const unsigned index = 26;
      static const FieldType *type;
    };

    struct MFVec3d
    {
      typedef SFVec3d Single;
      typedef vector<Single::Type> Type;
      typedef const Type &ArgType;
      typedef SimpleValue<Type> Value;
      static const unsigned index = 27;
      static const FieldType *type;
    };


    struct SFVec3f
    {
      typedef SFFloat::Type ElemType;
      static const unsigned size = 3;
      typedef BasicVector<ElemType, size> Type;
      typedef Type ArgType;
      typedef SimpleValue<Type> Value;
      static const unsigned index = 28;
      static const FieldType *type;
    };

    struct MFVec3f
    {
      typedef SFVec3f Single;
      typedef vector<Single::Type> Type;
      typedef const Type &ArgType;
      typedef SimpleValue<Type> Value;
      static const unsigned index = 29;
      static const FieldType *type;
    };

    const unsigned fieldTypeIndexRoof = 30;
  }
}

#endif // ARCHON_X3D_FIELD_TYPE_H
```

Listing: **archon/x3d/server/field_type.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
```



```
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#include <archon/x3d/server/field_type.H>

namespace Archon
{
  using namespace Utilities;

  namespace X3D
  {
    const FieldType *SFBool::type = 0;
    const FieldType *MFBool::type = 0;

    const FieldType *SFColor::type = 0;
    const FieldType *MFColor::type = 0;

    const FieldType *SFColorRGBA::type = 0;
    const FieldType *MFColorRGBA::type = 0;

    const FieldType *SFDouble::type = 0;
    const FieldType *MFDouble::type = 0;

    const FieldType *SFFloat::type = 0;
    const FieldType *MFFloat::type = 0;

    const FieldType *SFImage::type = 0;
    const FieldType *MFImage::type = 0;

    const FieldType *SFInt32::type = 0;
    const FieldType *MFInt32::type = 0;

    const FieldType *SFNode::type = 0;
    const FieldType *MFNode::type = 0;

    const FieldType *SFRotation::type = 0;
    const FieldType *MFRotation::type = 0;

    const FieldType *SFString::type = 0;
    const FieldType *MFString::type = 0;

    const FieldType *SFTime::type = 0;
    const FieldType *MFTime::type = 0;

    const FieldType *SFVec2d::type = 0;
    const FieldType *MFVec2d::type = 0;

    const FieldType *SFVec2f::type = 0;
    const FieldType *MFVec2f::type = 0;
```

```
const FieldType *SFVec3d::type = 0;
const FieldType *MFVec3d::type = 0;

const FieldType *SFVec3f::type = 0;
const FieldType *MFVec3f::type = 0;


void initializeFieldTypes()
{
  SFBool::type      = new FieldType(SFBool::index,      "SFBool");
  MFBool::type      = new FieldType(MFBool::index,      "MFBool",
       SFBool::type);

  SFColor::type     = new FieldType(SFColor::index,     "SFColor");
  MFColor::type     = new FieldType(MFColor::index,     "MFColor",
       SFColor::type);

  SFColorRGBA::type = new FieldType(SFColorRGBA::index, "SFColorRGBA");
  MFColorRGBA::type = new FieldType(MFColorRGBA::index, "MFColorRGBA",
       SFColorRGBA::type);

  SFDouble::type    = new FieldType(SFDouble::index,    "SFDouble");
  MFDouble::type    = new FieldType(MFDouble::index,    "MFDouble",
       SFDouble::type);

  SFFloat::type     = new FieldType(SFFloat::index,     "SFFloat");
  MFFloat::type     = new FieldType(MFFloat::index,     "MFFloat",
       SFFloat::type);

  SFImage::type     = new FieldType(SFImage::index,     "SFImage");
  MFImage::type     = new FieldType(MFImage::index,     "MFImage",
       SFImage::type);

  SFInt32::type     = new FieldType(SFInt32::index,     "SFInt32");
  MFInt32::type     = new FieldType(MFInt32::index,     "MFInt32",
       SFInt32::type);

  SFNode::type      = new FieldType(SFNode::index,      "SFNode");
  MFNode::type      = new FieldType(MFNode::index,      "MFNode",
       SFNode::type);

  SFRotation::type  = new FieldType(SFRotation::index,  "SFRotation");
  MFRotation::type  = new FieldType(MFRotation::index,  "MFRotation",
       SFRotation::type);

  SFString::type    = new FieldType(SFString::index,    "SFString");
  MFString::type    = new FieldType(MFString::index,    "MFString",
       SFString::type);

  SFTime::type      = new FieldType(SFTime::index,      "SFTime");
  MFTime::type      = new FieldType(MFTime::index,      "MFTime",
       SFTime::type);

  SFVec2d::type     = new FieldType(SFVec2d::index,     "SFVec2d");
  MFVec2d::type     = new FieldType(MFVec2d::index,     "MFVec2d",
       SFVec2d::type);
```



```
      SFVec2f::type     = new FieldType(SFVec2f::index,     "SFVec2f");
      MFVec2f::type     = new FieldType(MFVec2f::index,     "MFVec2f",
           SFVec2f::type);

      SFVec3d::type     = new FieldType(SFVec3d::index,     "SFVec3d");
      MFVec3d::type     = new FieldType(MFVec3d::index,     "MFVec3d",
           SFVec3d::type);

      SFVec3f::type     = new FieldType(SFVec3f::index,     "SFVec3f");
      MFVec3f::type     = new FieldType(MFVec3f::index,     "MFVec3f",
           SFVec3f::type);
    }
  }
}
```

Listing: **archon/x3d/server/font.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_FONT_H
#define ARCHON_X3D_FONT_H

#include <string>
#include <vector>

#include <archon/util/ref.H>

namespace Archon
{
  namespace X3D
  {
    using namespace std;
    using namespace Utilities;

    struct Server;

    /**
     * This class is responsible for locating font files and rendering
     * them into OpenGL textures. It is also responsible for keeping
     * track of the usage of those textures and arrange for the
     * destruction of them when appropriate.
```

```
 */
struct FontServer: virtual RefObjectBase
{
  struct Exception: virtual X3D::Exception {};
  struct ArgumentException: Exception, X3D::ArgumentException
  {
    ArgumentException(string l, string m): Utilities::Exception(l, m) {}
  };
  struct ResourceException: Exception, X3D::ResourceException
  {
    ResourceException() {}
    ResourceException(string l, string m): Utilities::Exception(l, m) {}
  };
  struct StateException: Exception, X3D::StateException
  {
    StateException(string l, string m): Utilities::Exception(l, m) {}
  };
  struct InternalException: Exception, X3D::InternalException
  {
    InternalException(string l, string m): Utilities::Exception(l, m) {}
  };

  FontServer(Ref<Server>, vector<string> fontDirs);

  const BackRef<Server> server;

private:
  vector<string> fontDirs;
};
```

```
namespace
{
  struct Glyph
  {
    /**
     * Size of layout bbox in units of standard line spacing.
     */
    Vector2 size;

    /**
     * Position of lower left corner of bbox of horizontal layout
     * in units of standard line spacing relative to the pen
     * position on the baseline.
     */
    Vector2 horizontalLowerLeft;

    /**
     * Pen advance for horizontal layout.
     */
    double horizontalAdvance;

    /**
     * Position of lower left corner of bbox of vertical layout in
```



```
     * units of standard line spacing relative to the pen position
     * on the baseline.
     */
    Vector2 verticalLowerLeft;

    /**
     * Pen advance for vertical layout.
     */
    double verticalAdvance;

    Vector2 textureLowerLeft;
    Vector2 textureUpperRight;

    Glyph(Vector2 size,
          Vector2 horizontalLowerLeft,
          double horizontalAdvance,
          Vector2 verticalUpperRight,
          double verticalAdvance,
          Vector2 textureLowerLeft,
          Vector2 textureUpperRight):
      size(size),
      horizontalLowerLeft(horizontalLowerLeft),
      horizontalAdvance(horizontalAdvance),
      verticalLowerLeft(verticalLowerLeft),
      verticalAdvance(verticalAdvance),
      textureLowerLeft(textureLowerLeft),
      textureUpperRight(textureUpperRight) {}
  };

  struct FontFace;

  struct FontList
  {
    Mutex mutex;

    static vector<string> fontPaths;
    static int pixelSize;

    struct Entry
    {
      string plainFilePath;
      int plainFileIndex;
      FontFace *plainFace;

      string boldFilePath;
      int boldFileIndex;
      FontFace *boldFace;

      string italicFilePath;
      int italicFileIndex;
      FontFace *italicFace;

      string bolditalicFilePath;
      int bolditalicFileIndex;
      FontFace *bolditalicFace;

      Entry():
        plainFace(0), boldFace(0),
```

```
    italicFace(0), bolditalicFace(0) {}
};

map<string, Entry> fontFamilies;

FT_Library library;

bool addFace(string path, int index, FT_Face face)
{
  string family = face->family_name;

  bool bold   = face->style_flags&FT_STYLE_FLAG_BOLD;
  bool italic = face->style_flags&FT_STYLE_FLAG_ITALIC;

  //string style = bold ? italic ? "bolditalic" : "bold" : italic ? "
       italic" : "plain";
  //cerr << "Found \"" << family << "\" " << style << "\n";

  /*
  cerr << "Style name        = " << face->style_name << "\n";
  cerr << "Number of glyphs  = " << face->num_faces << "\n";
  cerr << "Units per EM      = " << face->units_per_EM << "\n";
  cerr << "Height            = " << face->height << "\n";
  cerr << "Ascender          = " << face->ascender << "\n";
  cerr << "Descender         = " << face->descender << "\n";
  cerr << "Num fixed sizes   = " << face->num_fixed_sizes << "\n";
  cerr << "Num charmaps      = " << face->num_charmaps << "\n";
  if(face->face_flags&FT_FACE_FLAG_SCALABLE)
    cerr << "                   FT_FACE_FLAG_SCALABLE\n";
  if(face->face_flags&FT_FACE_FLAG_FIXED_SIZES)
    cerr << "                   FT_FACE_FLAG_FIXED_SIZES\n";
  if(face->face_flags&FT_FACE_FLAG_FIXED_WIDTH)
    cerr << "                   FT_FACE_FLAG_FIXED_WIDTH\n";
  if(face->face_flags&FT_FACE_FLAG_HORIZONTAL)
    cerr << "                   FT_FACE_FLAG_HORIZONTAL\n";
  if(face->face_flags&FT_FACE_FLAG_VERTICAL)
    cerr << "                   FT_FACE_FLAG_VERTICAL\n";
  if(face->face_flags&FT_FACE_FLAG_SFNT)
    cerr << "                   FT_FACE_FLAG_SFNT\n";
  if(face->face_flags&FT_FACE_FLAG_KERNING)
    cerr << "                   FT_FACE_FLAG_KERNING\n";
  if(face->face_flags&FT_FACE_FLAG_MULTIPLE_MASTERS)
    cerr << "                   FT_FACE_FLAG_MULTIPLE_MASTERS\n";
  if(face->face_flags&FT_FACE_FLAG_GLYPH_NAMES)
    cerr << "                   FT_FACE_FLAG_GLYPH_NAMES\n";
  if(face->face_flags&FT_FACE_FLAG_EXTERNAL_STREAM)
    cerr << "                   FT_FACE_FLAG_EXTERNAL_STREAM\n";
  if(face->face_flags&FT_FACE_FLAG_FAST_GLYPHS)
    cerr << "                   FT_FACE_FLAG_FAST_GLYPHS\n";
  */

  Entry &entry = fontFamilies[family];

  if(!bold && !italic)
  {
    if(entry.plainFilePath.size()) return false;
    entry.plainFilePath = path;
    entry.plainFileIndex = index;
```



```
  }
  else if(bold && !italic)
  {
    if(entry.boldFilePath.size()) return false;
    entry.boldFilePath = path;
    entry.boldFileIndex = index;
  }
  else if(!bold && italic)
  {
    if(entry.italicFilePath.size()) return false;
    entry.italicFilePath = path;
    entry.italicFileIndex = index;
  }
  else if(bold && italic)
  {
    if(entry.bolditalicFilePath.size()) return false;
    entry.bolditalicFilePath = path;
    entry.bolditalicFileIndex = index;
  }

  return true;
}

FontList()
{
  if(FT_Init_FreeType(&library))
    ARCHON_THROW1(ResourceException,
                  "Error initializing FreeType library");

  int n = 0;

  for(unsigned i=0; i<fontPaths.size(); ++i)
  {
    //cerr << "Searching for fonts in \"" << fontPaths[i] << "\"\n";
    vector<string> dirNames;
    try
    {
      dirNames = File::getDirNames(fontPaths[i]);
    }
    catch(File::ArgumentException &)
    {
      // ************************************   ISSUE A WARNING
           THROUGH THE SERVERS LOGGER ***************************
      continue;
    }
    for(unsigned j=0; j<dirNames.size(); ++j)
    {
      string name = dirNames[j];
      if(name == "." || name == "..") continue;
      string p = fontPaths[i]+"/"+name;
      if(File::Stat(p).getType() != File::Stat::type_regular)
           continue;

      FT_Face face;
      if(FT_New_Face(library, p.c_str(), 0, &face)) continue;

      //cerr << "Found font file \"" << p << "\"\n";
```

```
        if(addFace(p, 0, face)) ++n;

        unsigned m = face->num_faces;
        FT_Done_Face(face);

        for(unsigned k=1; k<m; ++k)
        {
          if(FT_New_Face(library, p.c_str(), k, &face)) break;
          if(addFace(p, k, face)) ++n;
          FT_Done_Face(face);
        }
      }
    }

    cerr << "Found " + Archon::Utilities::Text::toString(n) +
      " font faces in " + Archon::Utilities::Text::toString(fontPaths.
           size()) +
      " directories\n";
  }

  FontFace *getFace(string familyName, bool bold, bool italic);
  FontFace *getDefaultFace();
};

vector<string> FontList::fontPaths;
int FontList::pixelSize = 32;

struct FontFace
{
  Mutex mutex;

  double horizontalBottom;
  double horizontalTop;

  double verticalLeft;
  double verticalRight;

  vector<const Glyph *> glyphs;

  GLuint textureId;

  FontFace(string filePath, int faceIndex, int pixelSize)
  {
    const FontList *fontList = getFontList();

    FT_Face face;
    if(FT_New_Face(fontList->library, filePath.c_str(), faceIndex, &
         face))
      ARCHON_THROW1(ResourceException,
                    "Font file \"" + filePath + "\" is no longer
                         available");

    if(FT_Set_Pixel_Sizes(face, 0, pixelSize))
      ARCHON_THROW1(InternalException,
                    "Error setting pixel size of font face");

    FT_GlyphSlot slot = face->glyph;
```



```
    glyphs.resize(256);

    int imageWidth  = 16*pixelSize;
    int imageHeight = 16*pixelSize;
    Image texture(imageWidth, imageHeight, Image::components_la, 8);
         // Could be reduced to 1 bit per component

    for(int x=0; x<imageWidth; ++x)
      for(int y=0; y<imageHeight; ++y)
        texture.setPixel(x, y, 255u, 0u);

    /*
     * Minimum vertical and horizontal distance between glyphs
     * in texture mesured in number of texels. This spacing is
     * introduced to prevent the texels from one glyph to
     * "bleed" into the bbox of an adjacent glyph due to the
     * filtering done by OpenGL.
     */
    const int textureSpacing = 4;

    /*
     * Number of texels to expand the minimum bbox of the glyph
     * in the texture by to get the bbox to use for the actual
     * texture mapping. This of cause increases the size of the
     * rendered quad for the glyph. The reason for expanding the
     * minimum bbox is to allow the filtering done by OpenGL to
     * work correctly at the edges of the glyph. If there were
     * no padding then otherwise smooth edges of the glyph would
     * be cut off too sharply. Note that half the size of
     * 'textureSpacing' will usually be optimal.
     */
    const double texturePadding = textureSpacing/2.0;

    int xPos = textureSpacing;
    int yPos = textureSpacing;
    int maxHeight = 0;

    // Load first Unicode page into texture
    for(unsigned i=0; i<256; ++i)
    {
      if(FT_Load_Char(face, static_cast<unsigned char>(i),
           FT_LOAD_RENDER))
        continue;
      if(int(slot->bitmap.pixel_mode) != 2)
        ARCHON_THROW1(InternalException,
                      "Unsupported pixel data encoding in font");

      if(slot->bitmap.width+textureSpacing > imageWidth - xPos)
      {
        yPos += maxHeight+textureSpacing;
        xPos = textureSpacing;
        maxHeight = 0;
      }

      if(slot->bitmap.width+textureSpacing > imageWidth - xPos ||
         slot->bitmap.rows+textureSpacing > imageHeight - yPos)
        ARCHON_THROW1(InternalException,
                      "All glyphs won't fit inside texture");
```

```
        for(int x=0; x<slot->bitmap.width; ++x)
          for(int y=0; y<slot->bitmap.rows; ++y)
          {
            unsigned l = (static_cast<unsigned char *>(slot->bitmap.
                buffer))
              [x + y * slot->bitmap.width];
            texture.setPixel(xPos+x, imageHeight-1-yPos-y, 255, l);
          }

        double linesPerTexel = face->units_per_EM/double(face->height*
            pixelSize);

        horizontalBottom = 0;
        horizontalTop    = pixelSize*linesPerTexel;
        verticalLeft     = 0;
        verticalRight    = pixelSize*linesPerTexel;

        Vector2 horizontalLowerLeft;
        double horizontalAdvance;
        Vector2 verticalLowerLeft;
        double verticalAdvance;

        if(face->face_flags&FT_FACE_FLAG_HORIZONTAL)
        {
          horizontalLowerLeft = verticalLowerLeft =
            Vector2(slot->bitmap_left-texturePadding,
                    slot->bitmap_top-slot->bitmap.rows-texturePadding)*
            linesPerTexel;

          horizontalAdvance = slot->advance.x/64 * linesPerTexel;
          verticalAdvance   = 1;
        }
        else
        {
          horizontalLowerLeft = verticalLowerLeft =
            Vector2(slot->bitmap_left-texturePadding,
                    slot->bitmap_top-slot->bitmap.rows-texturePadding)*
            linesPerTexel;

          horizontalAdvance = 1;
          verticalAdvance   = slot->advance.y/64 * linesPerTexel;
        }

        glyphs[i] =
          new Glyph(Vector2(slot->bitmap.width + 2*texturePadding,
                            slot->bitmap.rows  + 2*texturePadding) *
                    linesPerTexel,
                    horizontalLowerLeft, horizontalAdvance,
                    verticalLowerLeft, verticalAdvance,
                    Vector2(double(xPos-texturePadding)/imageWidth,
                            double(imageHeight-1-yPos-slot->bitmap.rows-
                                texturePadding)/imageHeight),
                    Vector2(double(xPos+slot->bitmap.width+texturePadding
                        )/imageWidth,
                            double(imageHeight-1-yPos+texturePadding)/
                                imageHeight));
```



```
        xPos += slot->bitmap.width+textureSpacing;
        if(slot->bitmap.rows > maxHeight) maxHeight = slot->bitmap.rows;
      }

      {
        string family = face->family_name;
        bool bold   = face->style_flags&FT_STYLE_FLAG_BOLD;
        bool italic = face->style_flags&FT_STYLE_FLAG_ITALIC;
        string style = bold ? italic ? "bolditalic" : "bold" : italic ? "
            italic" : "plain";
        texture.save(family + " " + style + ".png");
      }

      FT_Done_Face(face);

      {
        int colorMode;
        switch(texture.getComponentSpecifier())
        {
        case Utilities::Image::components_l:    colorMode = GL_LUMINANCE
            ;       break;
        case Utilities::Image::components_la:   colorMode =
            GL_LUMINANCE_ALPHA; break;
        case Utilities::Image::components_rgb:  colorMode = GL_RGB;
            break;
        case Utilities::Image::components_rgba: colorMode = GL_RGBA;
            break;
        default:
          ARCHON_THROW1(InternalException,
                        "Unsuported color components");
        }

        int componentType;
        switch(texture.getBitsPerComponent())
        {
        case 8:  componentType = GL_UNSIGNED_BYTE;  break;
        case 16: componentType = GL_UNSIGNED_SHORT; break;
        default:
          ARCHON_THROW1(InternalException,
                        "Unsuported color component width");
        }

        glGenTextures(1, &textureId);
        glBindTexture(GL_TEXTURE_2D, textureId);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,GL_CLAMP);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

        glTexImage2D(GL_TEXTURE_2D, 0, colorMode, texture.getWidth(),
                     texture.getHeight(), 0, colorMode, componentType,
                     texture.getDataPtr());
      }
    }

    /**
     * \todo Must dynamically detect missing unicode pages and
```

```
     * create them on demand
     */
    const Glyph *getGlyph(uchar i)
    {
      Mutex::Lock l(mutex);
      if(i > 255) return 0;
      return glyphs[i];
    }

    /**
     * Get the default face of the default font family.
     *
     * \return Never null.
     */
    static FontFace *get()
    {
      FontList *fontList = getFontList();
      return fontList->getDefaultFace();
    }

    /**
     * Get the specified face of the specified font family.
     *
     * \return Null if the font face was not found.
     */
    static FontFace *get(string familyName, bool bold, bool italic)
    {
      FontList *fontList = getFontList();
      return fontList->getFace(familyName, bold, italic);
    }

  private:
    static FontList *getFontList()
    {
      static FontList fontList;
      return &fontList;
    }
  };
}

FontFace *FontList::getDefaultFace()
{
  FontFace *f = getFace("Times New Roman", false, false);
  if(f) return f;
  ARCHON_THROW1(InternalException,
                "Could not find default font face");
}

FontFace *FontList::getFace(string familyName, bool bold, bool italic)
{
  Mutex::Lock l(mutex);

  map<string, Entry>::iterator i = fontFamilies.find(familyName);
  if(i == fontFamilies.end()) return 0;

  if(!bold && !italic)
  {
    if(!i->second.plainFilePath.size()) return 0;
```



```
        if(!i->second.plainFace)
          i->second.plainFace =
            new FontFace(i->second.plainFilePath, i->second.plainFileIndex,
                 pixelSize);
        return i->second.plainFace;
      }

      if(bold && !italic)
      {
        if(!i->second.boldFilePath.size()) return 0;
        if(!i->second.boldFace)
          i->second.boldFace =
            new FontFace(i->second.boldFilePath, i->second.boldFileIndex,
                 pixelSize);
        return i->second.boldFace;
      }

      if(!bold && italic)
      {
        if(!i->second.italicFilePath.size()) return 0;
        if(!i->second.italicFace)
          i->second.italicFace =
            new FontFace(i->second.italicFilePath, i->second.italicFileIndex
                 , pixelSize);
        return i->second.italicFace;
      }

      if(!i->second.bolditalicFilePath.size()) return 0;
      if(!i->second.bolditalicFace)
        i->second.bolditalicFace =
          new FontFace(i->second.bolditalicFilePath, i->second.
               bolditalicFileIndex, pixelSize);
      return i->second.bolditalicFace;
    }



  }
}

#endif // ARCHON_X3D_FONT_H
```

Listing: **archon/x3d/server/font.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
```

```
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#include <archon/x3d/server/font.H>

#include <archon/x3d/server/server.H>

namespace Archon
{
  namespace X3D
  {
    FontServer::FontServer(Ref<Server> server, vector<string> fontDirs):
      server(server), fontDirs(fontDirs)
    {
      for(unsigned i=0; i<fontDirs.size(); ++i)
      {
        cerr << "FontDir: '" << fontDirs[i] << "'\n";
      }
    }
  }
}
```

Listing: archon/x3d/server/geometry.H

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_GEOMETRY_H
#define ARCHON_X3D_GEOMETRY_H

/*
 * \file geometry.H
 *
 *  X3D Geometry component
 */

#include <archon/x3d/server/geoprop.H>
#include <archon/x3d/server/texture.H>
#include <archon/x3d/server/appear.H>
```



```
namespace Archon
{
  namespace X3D
  {
    struct RenderConfig
    {
      /*
       * Subdivision1 Subdivision along the primary axis of the of
       * curved surfaces such as spheres, tori, cylinders and cones.
       */
      int subdivision1;

      /**
       * Subdivision2 Subdivision along the secondary axis.
       */
      int subdivision2;

      bool showNormals;

      Vector3 normalColor;
      double normalLength;

      RenderConfig(int subdivision1,
                   int subdivision2,
                   bool showNormals,
                   double normalLength):
        subdivision1(subdivision1),
        subdivision2(subdivision2),
        showNormals(showNormals),
        normalColor(1, 0, 0),
        normalLength(normalLength) {}
    };

    struct Shape;

    struct GeometryNode: virtual Node
    {
      static const NodeType *type;

      /**
       * Determine the distance along the ray from the ray origin to the
       * surface of this geometric object. The distance is mesured in
       * units of the ray direction vector length and is never
       * negative.
       *
       * Returns zero if the ray does not intersect the
       * geometry. Otherwise non-zero, and the returned value will
       * indicate which part of the geometry that was intersected.
       *
       * When zero is returned the 'dist' parameter is left
       * unchaged. Otherwise it is set to the distance in question.
       *
       * For solod objects like spheres this method shall always return
       * zero when the ray originates from a point in the interior.
       */
      virtual int intersect(const Math::Ray3 &, double &dist) const = 0;

      /**
```

```
   * Render the geometric structure using OpenGL primitives.  For
   * each vertex set the normal, the color and the texture
   * coordinate. The texture coordinate mey be skipped if the
   * 'texture' parameter is false. The color may be skipped in
   * which case the effective color is controlled by the material.
   *
   * The following GL parameters may not be assumed to have any
   * particular value and may be left changed: vertex normal,
   * vertex color, texture coordinate, polygon front face, color
   * material tracking, one/two sided light model, cull face
   * enable.
   *
   * \param texture If false the texture coordinates may be left
   * unspecified.
   *
   * \param color If false the current color of OpenGL must not be
   * updated.
   *
   * \param shape The Shape node that contains this geometry node
   * in the current thread of thraversal of the scene graph.
   */
  virtual void render(bool texture, bool color,
                      const Shape *shape,
                      const RenderConfig *) const = 0;

protected:
  GeometryNode() {}
};


struct Geometry3DNode: GeometryNode
{
  static const NodeType *type;

protected:
  Geometry3DNode() {}
};


/**
 * The Box node specifies a rectangular parallelepiped box
 * centred at the origin of the local coordinate system
 * and aligned with the local coordinate axes.
 * By default, the box measures 2 units in each dimension.
 *
 * Textures are applied individually to each face of the box.
 * On the front, back, right, and left faces of the box,
 * when viewed from the outside with the positive Y-axis pointing upwards
     ,
 * the texture is mapped onto each face with the same orientation
 * as if the image were displayed normally in 2D.
 * On the top face of the box, when viewed from above
 * and looking down the Y-axis toward the origin with the -Z-axis
 * as the view up direction, the texture is mapped onto the face
 * with the same orientation as if the image were displayed normally in 2
     D.
 * On the bottom face of the box, when viewed from below
 * looking up the Y-axis toward the origin with the +Z-axis
```



```
 * as the view up direction, the texture is mapped onto the face
 * with the same orientation as if the image were displayed normally in 2
     D.
 */
struct Box: Geometry3DNode
{
  static Ref<Node> instantiate(BackRef<ExecutionContext> c)
  {
    return new Box(c);
  }

  static const NodeType *type;
  const NodeType *getType() const { return type; }

  Box(BackRef<ExecutionContext> c): Node(c), size(2, 2, 2) {}

  const Vector3 &getSize() const { return size; }

  int intersect(const Math::Ray3 &, double &dist) const;
  void render(bool texture, bool color,
              const Shape *shape,
              const RenderConfig *) const;

private:
  friend void initializeGeometryComponent();

  Vector3 size;
};


/**
 * The Cone node specifies a cone which is centred in the local
     coordinate
 * system and whose central axis is aligned with the local Y-axis.
 * The bottomRadius field specifies the radius of the cone's base,
 * and the height field specifies the height of the cone from the centre
 * of the base to the apex. By default, the cone has a radius of 1.0
 * at the bottom and a height of 2.0, with its apex at y = height/2
 * and its bottom at y = -height/2.
 *
 * The side field specifies whether sides of the cone are created
 * and the bottom field specifies whether the bottom cap of the cone
 * is created.
 *
 * When a texture is applied to the sides of the cone, the texture wraps
 * counterclockwise (from above) starting at the back of the cone.
 * The texture has a vertical seam at the back in the X=0 plane, from
 * the apex (0, height/2, 0) to the point (0, -height/2, -bottomRadius).
 * For the bottom cap, a circle is cut out of the texture square centred
 * at (0, -height/2, 0) with dimensions (2 x bottomRadius) by
 * (2 x bottomRadius). The bottom cap texture appears right side up when
 * the top of the cone is rotated towards the -Z-axis.
 */
struct Cone: Geometry3DNode
{
  static Ref<Node> instantiate(BackRef<ExecutionContext> c)
  {
    return new Cone(c);
```

```
  }

  static const NodeType *type;
  const NodeType *getType() const { return type; }

  Cone(BackRef<ExecutionContext> c):
    Node(c), bottomRadius(1),
    height(2), side(true), bottom(true) {}

  double getBottomRadius() const { return bottomRadius; }
  double getHeight()       const { return height;       }
  bool   getSide()         const { return side;         }
  bool   getBottom()       const { return bottom;       }

  int intersect(const Math::Ray3 &, double &dist) const;
  void render(bool texture, bool color,
              const Shape *shape,
              const RenderConfig *) const;

private:
  friend void initializeGeometryComponent();

  double bottomRadius;
  double height;
  bool   side;
  bool   bottom;
};


/**
 * The Cylinder node specifies a capped cylinder centred at the origin
 * of the local coordinate system and with a central axis oriented along
 * the local Y-axis. The radius field specifies the radius of
 * the cylinder and the height field specifies the height of the cylinder
 * along the central axis. The default height is 2 and the default radius
 * is 1.
 *
 * When a texture is applied to a cylinder, it is applied differently
 * to the sides, top, and bottom. On the sides, the texture wraps
 * counterclockwise (from above) starting at the back of the cylinder.
 * The texture has a vertical seam at the back, intersecting the X=0
     plane.
 * For the top and bottom caps, a circle is cut out of the unit texture
 * squares centred at (0, +/- height/2, 0) with dimensions 2 x radius
 * by 2 x radius. The top texture appears right side up when the top of
 * the cylinder is tilted toward the +Z-axis, and the bottom texture
 * appears right side up when the top of the cylinder is tilted toward
 * the -Z-axis.
 */
struct Cylinder: Geometry3DNode
{
  static Ref<Node> instantiate(BackRef<ExecutionContext> c)
  {
    return new Cylinder(c);
  }

  static const NodeType *type;
  const NodeType *getType() const { return type; }
```



```
  Cylinder(BackRef<ExecutionContext> c):
    Node(c), height(2), radius(1),
    side(true), top(true), bottom(true) {}

  double getHeight() const { return height; }
  double getRadius() const { return radius; }
  bool   getSide()   const { return side;   }
  bool   getTop()    const { return top;    }
  bool   getBottom() const { return bottom; }

  int intersect(const Math::Ray3 &, double &dist) const;
  void render(bool texture, bool color,
              const Shape *shape,
              const RenderConfig *) const;

private:
  friend void initializeGeometryComponent();

  double height;
  double radius;
  bool   side;
  bool   top;
  bool   bottom;
};


struct IndexedLineSet: Geometry3DNode
{
  static Ref<Node> instantiate(BackRef<ExecutionContext> c)
  {
    return new IndexedLineSet(c);
  }

  static const NodeType *type;
  const NodeType *getType() const { return type; }

  IndexedLineSet(BackRef<ExecutionContext> c):
    Node(c), colorPerVertex(true), lineWidth(1) {}

  Ref<ColorNode>      getColor() const { return color; }
  Ref<CoordinateNode> getCoord() const { return coord; }
  vector<int>         getColorIndex() const { return colorIndex; }
  bool                getColorPerVertex() const { return colorPerVertex
      ; }
  vector<int>         getCoordIndex() const { return coordIndex; }
  double              getLineWidth() const { return lineWidth; }

  int intersect(const Math::Ray3 &, double &dist) const { return 0; }
  void render(bool texture, bool color,
              const Shape *shape,
              const RenderConfig *) const;

private:
  friend void initializeGeometryComponent();

  Ref<ColorNode>      color;          // [in,out] NULL [X3DColorNode]
```

```
  Ref<CoordinateNode> coord;          // [in,out] NULL [X3DCoordinateNode
       ]
  vector<int>         colorIndex;     // []       []   [0,inf) or -1
  bool                colorPerVertex; // []       TRUE
  vector<int>         coordIndex;     // []       []   [0,inf) or -1
  double              lineWidth;      // []       1    [1,inf)

  EventSource colorChanged;
  EventSource coordChanged;

  Time colorStamp;
  Time coordStamp;
  Time colorIndexStamp;
  Time coordIndexStamp;
};


struct PointSet: Geometry3DNode
{
  static Ref<Node> instantiate(BackRef<ExecutionContext> c)
  {
    return new PointSet(c);
  }

  static const NodeType *type;
  const NodeType *getType() const { return type; }

  PointSet(BackRef<ExecutionContext> c): Node(c) {}

  Ref<ColorNode>      getColor() const { return color; }
  Ref<CoordinateNode> getCoord() const { return coord; }

  int intersect(const Math::Ray3 &, double &dist) const { return 0; }
  void render(bool texture, bool color,
              const Shape *shape,
              const RenderConfig *) const;

private:
  friend void initializeGeometryComponent();

  Ref<ColorNode>      color; // [in,out] NULL [X3DColorNode]
  Ref<CoordinateNode> coord; // [in,out] NULL [X3DCoordinateNode]

  EventSource colorChanged;
  EventSource coordChanged;

  Time colorStamp;
  Time coordStamp;
};


/**
 * A sphere is centered in the origin of the local coordinate
 * system. The north pole of the sphere is in the direction of the
 * positive y-axis (upwards in the canonical view). A texture is
 * applied to the sphere by taking the polar coordinates from the
 * surface of the sphere as the texture coordinates (scaled
 * apropriately). The latitude or elevation angle plays the role
```



```
 * of the texture y-coordinate such that the north pole
 * corresponds to the top of the texture and the south pole
 * corresponds to the bottom. The logitude or azimuth angle plays
 * the role of the texture x-coordinate. The zero meridian is the
 * meridian that intersects the negative z-axis (ahead in the
 * canonical view). This corresponds to the left and right edge of
 * the texture. The texture is wraped counter-clockwise around the
 * sphere when seen from above.
 */
struct Sphere: Geometry3DNode
{
  static Ref<Node> instantiate(BackRef<ExecutionContext> c)
  {
    return new Sphere(c);
  }

  static const NodeType *type;
  const NodeType *getType() const { return type; }

  Sphere(BackRef<ExecutionContext> c): Node(c), radius(1) {}

  double getRadius() const { return radius; }

  int intersect(const Math::Ray3 &, double &dist) const;
  void render(bool texture, bool color,
              const Shape *shape,
              const RenderConfig *) const;

private:
  friend void initializeGeometryComponent();

  double radius;
};

struct ComposedGeometryNode: Geometry3DNode
{
  static const NodeType *type;

  bool getCCW()            const { return ccw; }
  bool getColorPerVertex() const { return colorPerVertex; }
  bool getNormalPerVertex() const { return normalPerVertex; }
  bool getSolid()          const { return solid; }

  Ref<const CoordinateNode>        getCoord()    const { return coord; }
  Ref<      CoordinateNode>        getCoord()          { return coord; }
  Ref<const ColorNode>             getColor()    const { return color; }
  Ref<      ColorNode>             getColor()          { return color; }
  Ref<const NormalNode>            getNormal()   const { return normal; }
  Ref<      NormalNode>            getNormal()         { return normal; }
  Ref<const TextureCoordinateNode> getTexCoord() const { return texCoord
       ; }
  Ref<      TextureCoordinateNode> getTexCoord()       { return texCoord
       ; }

protected:
  ComposedGeometryNode():
    ccw(true),
    colorPerVertex(true),
```

```
    normalPerVertex(true),
    solid(true) {}

  friend class Viewer;
  friend void initializeGeometryComponent();

  bool ccw;
  bool colorPerVertex;
  bool normalPerVertex;
  bool solid;

  Ref<CoordinateNode>        coord;
  Ref<ColorNode>             color;
  Ref<NormalNode>            normal;
  Ref<TextureCoordinateNode> texCoord;

  EventSource colorChanged;
  EventSource coordChanged;
  EventSource normalChanged;
  EventSource texCoordChanged;

  Time colorStamp;
  Time coordStamp;
  Time normalStamp;
  Time texCoordStamp;
};


struct ElevationGrid: ComposedGeometryNode
{
  static Ref<Node> instantiate(BackRef<ExecutionContext> c)
  {
    return new ElevationGrid(c);
  }

  static const NodeType *type;
  const NodeType *getType() const { return type; }

  ElevationGrid(BackRef<ExecutionContext> c):
    Node(c), creaseAngle(0),
    xDimension(0), xSpacing(1), zDimension(0), zSpacing(1) {}

  double getCreaseAngle()           const { return creaseAngle; }
  const vector<double> &getHeight() const { return height;      }
  int    getXDimension()            const { return xDimension;  }
  double getXSpacing()              const { return xSpacing;    }
  int    getZDimension()            const { return zDimension;  }
  double getZSpacing()              const { return zSpacing;    }

  int intersect(const Math::Ray3 &, double &dist) const;
  void render(bool texture, bool color,
              const Shape *shape,
              const RenderConfig *) const;

private:
  friend void initializeGeometryComponent();

  double         creaseAngle;
```



```
  vector<double> height;
  int            xDimension;
  double         xSpacing;
  int            zDimension;
  double         zSpacing;

  Time heightStamp;
};


struct IndexedFaceSet: ComposedGeometryNode
{
  static Ref<Node> instantiate(BackRef<ExecutionContext> c)
  {
    return new IndexedFaceSet(c);
  }

  static const NodeType *type;
  const NodeType *getType() const { return type; }

  IndexedFaceSet(BackRef<ExecutionContext> c):
    Node(c), convex(true), creaseAngle(0) {}

  bool getConvex()           const { return convex; }
  double getCreaseAngle()    const { return creaseAngle; }

  // -1 seperates faces
  // Indices into 'coord'
  const vector<int> &getCoordIndex()    const { return coordIndex; }
  // Indices into 'color'
  const vector<int> &getColorIndex()    const { return colorIndex; }
  // Indices into 'normal'
  const vector<int> &getNormalIndex()   const { return normalIndex; }
  // Indices into 'texCoord'
  const vector<int> &getTexCoordIndex() const { return texCoordIndex; }

  int intersect(const Math::Ray3 &, double &dist) const;
  void render(bool texture, bool color,
              const Shape *shape,
              const RenderConfig *) const;

private:
  friend void initializeGeometryComponent();

  bool convex;
  double creaseAngle;

  vector<int>  coordIndex;
  vector<int>  colorIndex;
  vector<int>  normalIndex;
  vector<int>  texCoordIndex;

  Time colorIndexStamp;
  Time coordIndexStamp;
  Time normalIndexStamp;
  Time texCoordIndexStamp;
};
```

```
  struct ShapeNode: virtual ChildNode
  {
    static const NodeType *type;

  protected:
    ShapeNode() {}
  };


  struct Shape: ShapeNode, BoundedObject
  {
    static Ref<Node> instantiate(BackRef<ExecutionContext> c)
    {
      return new Shape(c);
    }

    static const NodeType *type;
    const NodeType *getType() const { return type; }

    Shape(BackRef<ExecutionContext> c): Node(c), drawCache(0) {}

    Ref<const AppearanceNode> getAppearance() const { return appearance; }
    Ref<      AppearanceNode> getAppearance()       { return appearance; }
    Ref<const GeometryNode> getGeometry() const { return geometry; }
    Ref<      GeometryNode> getGeometry()       { return geometry; }

    struct DrawCache
    {
      virtual ~DrawCache() {}

      /**
       * Called by the shape nodes destructor to indicate that the
       * cache is no longer needed.
       */
      virtual void dispose() = 0;
    };

    DrawCache *drawCache;

    virtual ~Shape()
    {{ // The extra scope is needed to work around gcc3.2 bug #8287
      if(drawCache) drawCache->dispose();
    }}

  private:
    friend void initializeGeometryComponent();

    Ref<AppearanceNode> appearance;
    Ref<GeometryNode>   geometry;

    EventSource appearanceChanged;
    EventSource geometryChanged;

    Time appearanceStamp;
    Time geometryStamp;
  };
}
```



```
}

#endif // ARCHON_X3D_GEOMETRY_H
```

Listing: **archon/x3d/server/geometry.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

/*
 * \file geometry.C
 *
 *  X3D Geometry component
 */

#include <GL/gl.h>

#include <archon/x3d/server/field_type.H>
#include <archon/x3d/server/field.H>
#include <archon/x3d/server/geometry.H>

namespace Archon
{
  namespace X3D
  {
    extern long nodeTypeIndexEnumerator;

    const NodeType *GeometryNode::type = 0;
    const NodeType *Geometry3DNode::type = 0;

    const NodeType *Box::type = 0;
    const NodeType *Cone::type = 0;
    const NodeType *Cylinder::type = 0;
    const NodeType *IndexedLineSet::type = 0;
    const NodeType *PointSet::type = 0;
    const NodeType *Sphere::type = 0;

    const NodeType *ComposedGeometryNode::type = 0;
    const NodeType *ElevationGrid::type = 0;
    const NodeType *IndexedFaceSet::type = 0;

    const NodeType *ShapeNode::type = 0;
```

```
const NodeType *Shape::type = 0;

void initializeGeometryComponent()
{
  vector<const FieldBase *> fields;


  // GeometryNode

  GeometryNode::type =
    NodeType::newAbstract("GeometryNode", false, 0,
                          Node::type);


  // Geometry3DNode

  Geometry3DNode::type =
    NodeType::newAbstract("Geometry3DNode", false, 0,
                          GeometryNode::type);


  // Box

  fields.push_back(newPrivateField("size", SFVec3f::type,
                                   &Box::size));
  Box::type =
    NodeType::newConcrete("Box", Box::instantiate, &fields,
                          Geometry3DNode::type);
  fields.clear();


  // Cone

  fields.push_back(newPrivateField("bottomRadius", SFFloat::type,
                                   &Cone::bottomRadius));
  fields.push_back(newPrivateField("height", SFFloat::type,
                                   &Cone::height));
  fields.push_back(newPrivateField("side", SFBool::type,
                                   &Cone::side));
  fields.push_back(newPrivateField("bottom", SFBool::type,
                                   &Cone::bottom));
  Cone::type =
    NodeType::newConcrete("Cone", Cone::instantiate, &fields,
                          Geometry3DNode::type);
  fields.clear();


  // Cylinder

  fields.push_back(newPrivateField("bottom", SFBool::type,
                                   &Cylinder::bottom));
  fields.push_back(newPrivateField("radius", SFFloat::type,
                                   &Cylinder::radius));
  fields.push_back(newPrivateField("height", SFFloat::type,
                                   &Cylinder::height));
  fields.push_back(newPrivateField("side", SFBool::type,
                                   &Cylinder::side));
  fields.push_back(newPrivateField("top", SFBool::type,
```



```
                                   &Cylinder::top));
  Cylinder::type =
    NodeType::newConcrete("Cylinder", Cylinder::instantiate, &fields,
                          Geometry3DNode::type);
  fields.clear();


  // IndexedLineSet

  fields.push_back(newExposedField("color",
                                   &IndexedLineSet::color,
                                   &IndexedLineSet::colorChanged,
                                   &IndexedLineSet::colorStamp));
  fields.push_back(newExposedField("coord",
                                   &IndexedLineSet::coord,
                                   &IndexedLineSet::coordChanged,
                                   &IndexedLineSet::coordStamp));
  fields.push_back(newPrivateField("colorIndex", MFInt32::type,
                                   &IndexedLineSet::colorIndex));
  fields.push_back(newEventIn("set_colorIndex", MFInt32::type,
                              &IndexedLineSet::colorIndex,
                              &IndexedLineSet::colorIndexStamp,
                              FieldBase::NOVARIANT));
  fields.push_back(newPrivateField("colorPerVertex", SFBool::type,
                                   &IndexedLineSet::colorPerVertex));
  fields.push_back(newPrivateField("coordIndex", MFInt32::type,
                                   &IndexedLineSet::coordIndex));
  fields.push_back(newEventIn("set_coordIndex", MFInt32::type,
                              &IndexedLineSet::coordIndex,
                              &IndexedLineSet::coordIndexStamp,
                              FieldBase::NOVARIANT));
  fields.push_back(newPrivateField("lineWidth", SFFloat::type,
                                   &IndexedLineSet::lineWidth));
  IndexedLineSet::type =
    NodeType::newConcrete("IndexedLineSet", IndexedLineSet::instantiate
         , &fields,
                          Geometry3DNode::type);
  fields.clear();



  // PointSet

  fields.push_back(newExposedField("color",
                                   &PointSet::color,
                                   &PointSet::colorChanged,
                                   &PointSet::colorStamp));
  fields.push_back(newExposedField("coord",
                                   &PointSet::coord,
                                   &PointSet::coordChanged,
                                   &PointSet::coordStamp));
  PointSet::type =
    NodeType::newConcrete("PointSet", PointSet::instantiate, &fields,
                          Geometry3DNode::type);
  fields.clear();


  // Sphere
```

```
fields.push_back(newPrivateField("radius", SFFloat::type,
                                &Sphere::radius));
Sphere::type =
  NodeType::newConcrete("Sphere", Sphere::instantiate, &fields,
                       Geometry3DNode::type);
fields.clear();


// ComposedGeometryNode

fields.push_back(newExposedField("color",
                                &ComposedGeometryNode::color,
                                &ComposedGeometryNode::colorChanged,
                                &ComposedGeometryNode::colorStamp));
fields.push_back(newExposedField("coord",
                                &ComposedGeometryNode::coord,
                                &ComposedGeometryNode::coordChanged,
                                &ComposedGeometryNode::coordStamp));
fields.push_back(newExposedField("normal",
                                &ComposedGeometryNode::normal,
                                &ComposedGeometryNode::normalChanged,
                                &ComposedGeometryNode::normalStamp));
fields.push_back(newExposedField("texCoord",
                                &ComposedGeometryNode::texCoord,
                                &ComposedGeometryNode::texCoordChanged
                                     ,
                                &ComposedGeometryNode::texCoordStamp
                                     ));
fields.push_back(newPrivateField("ccw", SFBool::type,
                                &ComposedGeometryNode::ccw));
fields.push_back(newPrivateField("colorPerVertex", SFBool::type,
                                &ComposedGeometryNode::colorPerVertex
                                     ));
fields.push_back(newPrivateField("normalPerVertex", SFBool::type,
                                &ComposedGeometryNode::normalPerVertex
                                     ));
fields.push_back(newPrivateField("solid", SFBool::type,
                                &ComposedGeometryNode::solid));
ComposedGeometryNode::type =
  NodeType::newAbstract("ComposedGeometryNode", false, &fields,
                       Geometry3DNode::type);
fields.clear();


// IndexedFaceSet

fields.push_back(newPrivateField("convex", SFBool::type,
                                &IndexedFaceSet::convex));
fields.push_back(newPrivateField("colorIndex", MFInt32::type,
                                &IndexedFaceSet::colorIndex));
fields.push_back(newEventIn("set_colorIndex", MFInt32::type,
                           &IndexedFaceSet::colorIndex,
                           &IndexedFaceSet::colorIndexStamp,
                           FieldBase::NOVARIANT));
fields.push_back(newPrivateField("coordIndex", MFInt32::type,
                                &IndexedFaceSet::coordIndex));
fields.push_back(newEventIn("set_coordIndex", MFInt32::type,
```



```
                           &IndexedFaceSet::coordIndex,
                           &IndexedFaceSet::coordIndexStamp,
                           FieldBase::NOVARIANT));
fields.push_back(newPrivateField("creaseAngle", SFFloat::type,
                                &IndexedFaceSet::creaseAngle));
fields.push_back(newPrivateField("normalIndex", MFInt32::type,
                                &IndexedFaceSet::normalIndex));
fields.push_back(newEventIn("set_normalIndex", MFInt32::type,
                           &IndexedFaceSet::normalIndex,
                           &IndexedFaceSet::normalIndexStamp,
                           FieldBase::NOVARIANT));
fields.push_back(newPrivateField("texCoordIndex", MFInt32::type,
                                &IndexedFaceSet::texCoordIndex));
fields.push_back(newEventIn("set_texCoordIndex", MFInt32::type,
                           &IndexedFaceSet::texCoordIndex,
                           &IndexedFaceSet::texCoordIndexStamp,
                           FieldBase::NOVARIANT));
IndexedFaceSet::type =
  NodeType::newConcrete("IndexedFaceSet", IndexedFaceSet::instantiate
       , &fields,
                       ComposedGeometryNode::type);
fields.clear();


// ElevationGrid

fields.push_back(newPrivateField("creaseAngle", SFFloat::type,
                                &ElevationGrid::creaseAngle));
fields.push_back(newPrivateField("height", MFFloat::type,
                                &ElevationGrid::height));
fields.push_back(newEventIn("set_height", MFFloat::type,
                           &ElevationGrid::height,
                           &ElevationGrid::heightStamp,
                           FieldBase::NOVARIANT));
fields.push_back(newPrivateField("xDimension", SFInt32::type,
                                &ElevationGrid::xDimension));
fields.push_back(newPrivateField("xSpacing", SFFloat::type,
                                &ElevationGrid::xSpacing));
fields.push_back(newPrivateField("zDimension", SFInt32::type,
                                &ElevationGrid::zDimension));
fields.push_back(newPrivateField("zSpacing", SFFloat::type,
                                &ElevationGrid::zSpacing));
ElevationGrid::type =
  NodeType::newConcrete("ElevationGrid", ElevationGrid::instantiate, &
       fields,
                       ComposedGeometryNode::type);
fields.clear();


// ShapeNode

ShapeNode::type =
  NodeType::newAbstract("ShapeNode", false, 0,
                       ChildNode::type);


// Shape
```

```
  fields.push_back(newExposedField("appearance",
                                   &Shape::appearance,
                                   &Shape::appearanceChanged,
                                   &Shape::appearanceStamp));
  fields.push_back(newExposedField("geometry",
                                   &Shape::geometry,
                                   &Shape::geometryChanged,
                                   &Shape::geometryStamp));
  Shape::type =
    NodeType::newConcrete("Shape", Shape::instantiate, &fields,
                          ShapeNode::type, BoundedObject::type);
  fields.clear();
}

int Box::intersect(const Math::Ray3 &ray, double &dist) const
{
  Math::Box3 box(size);
  return Math::intersect(ray, box, dist);
}

/**
 * \todo Do it!
 */
int Cone::intersect(const Math::Ray3 &, double &dist) const
{
  return false;
}

/**
 * \todo Is it correct to considder a cylinder with missing parts
 * as non-solid and a cylinder with all parts enabled as a solid
 * object? (I think I saw a confirmation of this somewhere in the
 * spec -KKS)
 */
int Cylinder::intersect(const Math::Ray3 &ray, double &dist) const
{
  bool enterOnly = side & top & bottom;
  return Math::intersectCylinder(ray, height, radius, dist, side,
                                 top, bottom, enterOnly) ? 1 : 0;
}

int Sphere::intersect(const Math::Ray3 &ray, double &dist) const
{
  Math::Sphere3 sphere(radius);
  return Math::intersect(ray, sphere, dist) ? 1 : 0;
}

/**
 * \todo Do it!
 */
int ElevationGrid::intersect(const Math::Ray3 &, double &dist) const
{
  return false;
}

/**
 * \todo Do it!
 */
```



```
int IndexedFaceSet::intersect(const Math::Ray3 &, double &dist) const
{
  return false;
}


void renderNormals(const vector<Vector3> &n, const RenderConfig *c)
{
  glDisable(GL_LIGHTING);
  glDisable(GL_TEXTURE_2D);
  glColor3f(c->normalColor[0], c->normalColor[1], c->normalColor[2]);
  glBegin(GL_LINES);
  unsigned i = 0;
  while(i<n.size())
  {
    Vector3 v = n[i++];
    glVertex3d(v[0], v[1], v[2]);
    v += n[i++] * c->normalLength;
    glVertex3d(v[0], v[1], v[2]);
  }
  glEnd();
}


void Box::render(bool texture, bool color,
                 const Shape *shape,
                 const RenderConfig *config) const
{
  glFrontFace(GL_CCW);
  glDisable(GL_COLOR_MATERIAL);
  glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, 0);
  glEnable(GL_CULL_FACE);

  const double x = size[0]/2;
  const double y = size[1]/2;
  const double z = size[2]/2;

  glBegin(GL_QUADS);

  // Left
  glNormal3d(-1,  0,  0);
  if(texture) glTexCoord2d(0, 0);
  glVertex3d(-x, -y, -z);
  if(texture) glTexCoord2d(1, 0);
  glVertex3d(-x, -y,  z);
  if(texture) glTexCoord2d(1, 1);
  glVertex3d(-x,  y,  z);
  if(texture) glTexCoord2d(0, 1);
  glVertex3d(-x,  y, -z);

  // Right
  glNormal3d( 1,  0,  0);
  if(texture) glTexCoord2d(0, 0);
  glVertex3d( x, -y,  z);
  if(texture) glTexCoord2d(1, 0);
  glVertex3d( x, -y, -z);
  if(texture) glTexCoord2d(1, 1);
  glVertex3d( x,  y, -z);
```

```
    if(texture) glTexCoord2d(0, 1);
    glVertex3d( x,  y,  z);

    // Bottom
    glNormal3d( 0, -1,  0);
    if(texture) glTexCoord2d(0, 0);
    glVertex3d(-x, -y, -z);
    if(texture) glTexCoord2d(1, 0);
    glVertex3d( x, -y, -z);
    if(texture) glTexCoord2d(1, 1);
    glVertex3d( x, -y,  z);
    if(texture) glTexCoord2d(0, 1);
    glVertex3d(-x, -y,  z);

    // Top
    glNormal3d( 0,  1,  0);
    if(texture) glTexCoord2d(0, 0);
    glVertex3d(-x,  y,  z);
    if(texture) glTexCoord2d(1, 0);
    glVertex3d( x,  y,  z);
    if(texture) glTexCoord2d(1, 1);
    glVertex3d( x,  y, -z);
    if(texture) glTexCoord2d(0, 1);
    glVertex3d(-x,  y, -z);

    // Back
    glNormal3d( 0,  0, -1);
    if(texture) glTexCoord2d(0, 0);
    glVertex3d( x, -y, -z);
    if(texture) glTexCoord2d(1, 0);
    glVertex3d(-x, -y, -z);
    if(texture) glTexCoord2d(1, 1);
    glVertex3d(-x,  y, -z);
    if(texture) glTexCoord2d(0, 1);
    glVertex3d( x,  y, -z);

    // Front
    glNormal3d( 0,  0,  1);
    if(texture) glTexCoord2d(0, 0);
    glVertex3d(-x, -y,  z);
    if(texture) glTexCoord2d(1, 0);
    glVertex3d( x, -y,  z);
    if(texture) glTexCoord2d(1, 1);
    glVertex3d( x,  y,  z);
    if(texture) glTexCoord2d(0, 1);
    glVertex3d(-x,  y,  z);

    glEnd();

    if(config->showNormals)
    {
      vector<Vector3> normals;

      // Left
      normals.push_back(Vector3(-x, -y, -z));
      normals.push_back(Vector3(-1,  0,  0));
      normals.push_back(Vector3(-x, -y,  z));
      normals.push_back(Vector3(-1,  0,  0));
```



```
      normals.push_back(Vector3(-x,  y,  z));
      normals.push_back(Vector3(-1,  0,  0));
      normals.push_back(Vector3(-x,  y, -z));
      normals.push_back(Vector3(-1,  0,  0));

      // Right
      normals.push_back(Vector3( x, -y,  z));
      normals.push_back(Vector3( 1,  0,  0));
      normals.push_back(Vector3( x, -y, -z));
      normals.push_back(Vector3( 1,  0,  0));
      normals.push_back(Vector3( x,  y, -z));
      normals.push_back(Vector3( 1,  0,  0));
      normals.push_back(Vector3( x,  y,  z));
      normals.push_back(Vector3( 1,  0,  0));

      // Bottom
      normals.push_back(Vector3(-x, -y, -z));
      normals.push_back(Vector3( 0, -1,  0));
      normals.push_back(Vector3( x, -y, -z));
      normals.push_back(Vector3( 0, -1,  0));
      normals.push_back(Vector3( x, -y,  z));
      normals.push_back(Vector3( 0, -1,  0));
      normals.push_back(Vector3(-x, -y,  z));
      normals.push_back(Vector3( 0, -1,  0));

      // Top
      normals.push_back(Vector3(-x,  y,  z));
      normals.push_back(Vector3( 0,  1,  0));
      normals.push_back(Vector3( x,  y,  z));
      normals.push_back(Vector3( 0,  1,  0));
      normals.push_back(Vector3( x,  y, -z));
      normals.push_back(Vector3( 0,  1,  0));
      normals.push_back(Vector3(-x,  y, -z));
      normals.push_back(Vector3( 0,  1,  0));

      // Back
      normals.push_back(Vector3( x, -y, -z));
      normals.push_back(Vector3( 0,  0, -1));
      normals.push_back(Vector3(-x, -y, -z));
      normals.push_back(Vector3( 0,  0, -1));
      normals.push_back(Vector3(-x,  y, -z));
      normals.push_back(Vector3( 0,  0, -1));
      normals.push_back(Vector3( x,  y, -z));
      normals.push_back(Vector3( 0,  0, -1));

      // Front
      normals.push_back(Vector3(-x, -y,  z));
      normals.push_back(Vector3( 0,  0,  1));
      normals.push_back(Vector3( x, -y,  z));
      normals.push_back(Vector3( 0,  0,  1));
      normals.push_back(Vector3( x,  y,  z));
      normals.push_back(Vector3( 0,  0,  1));
      normals.push_back(Vector3(-x,  y,  z));
      normals.push_back(Vector3( 0,  0,  1));

      renderNormals(normals, config);
    }
  }
```

```
void Cone::render(bool texture, bool color,
                  const Shape *shape,
                  const RenderConfig *config) const
{
  glFrontFace(GL_CW);
  glDisable(GL_COLOR_MATERIAL);
  if(side&bottom)
  {
    glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, 0);
    glEnable(GL_CULL_FACE);
  }
  else
  {
    glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, 1);
    glDisable(GL_CULL_FACE);
  }

  const int heightSteps = config->subdivision2;  // Eleveation /
       lattitude
  const int thetaSteps = config->subdivision1; // Azimuth / longitude

  const double r = bottomRadius;
  const double h = height;

  vector<Vector2> vl(thetaSteps);

  vector<Vector3> normals;
  const bool showNormals = config->showNormals;

  // Bottom
  if(bottom)
  {
    glBegin(GL_POLYGON);
    glNormal3d(0, -1, 0);
    const double y = h/-2;
    for(int i=0; i<thetaSteps; ++i)
    {
      const double theta = 2 * M_PI / thetaSteps * i;

      Vector2 v(-sin(theta), -cos(theta));
      if(texture) glTexCoord2d(v[0]/2+0.5, v[1]/2+0.5);
      vl[i] = v *= r;
      glVertex3d(v[0], y, v[1]);
    }
    glEnd();
    if(showNormals) for(int i=0; i<thetaSteps; ++i)
    {
      normals.push_back(Vector3(vl[i][0], y, vl[i][1]));
      normals.push_back(Vector3(0, -1, 0));
    }
  }
  else
  {
    for(int i=0; i<thetaSteps; ++i)
    {
      const double theta = 2 * M_PI / thetaSteps * i;
      vl[i] = Vector2(-sin(theta)*r, -cos(theta)*r);
```



```
    }
  }

  if(side)
  {
    Vector2 n(h, r);
    n.normalize();

    double prevTexT = 0;
    double prevY = h/-2;

    Vector2 v;

    // Middle quad strips
    for(int i=1; i<=heightSteps; ++i)
    {
      glBegin(GL_QUAD_STRIP);

      const Vector2 w = vl[0];

      const double texT = 1.0 / heightSteps * i;
      const double y = h * (texT - 0.5);

      for(int j=0; j<thetaSteps; ++j)
      {
        const double texS = 1.0 / thetaSteps * j;
        const double theta = 2 * M_PI * texS;

        double x = -sin(theta);
        double z = -cos(theta);

        glNormal3d(x*n[0], n[1], z*n[0]);

        if(texture) glTexCoord2d(texS, prevTexT);
        v = vl[j];
        glVertex3d(v[0], prevY, v[1]);
        if(showNormals)
        {
          normals.push_back(Vector3(v[0], prevY, v[1]));
          normals.push_back(Vector3(x*n[0], n[1], z*n[0]));
        }

        if(texture) glTexCoord2d(texS, texT);
        v = Vector2(x, z);
        v *= r * (1 - texT);
        glVertex3d(v[0], y, v[1]);
        vl[j] = v;
      }

      glNormal3d(0, n[1], -n[0]);

      if(texture) glTexCoord2d(1, prevTexT);
      glVertex3d(w[0], prevY, w[1]);

      if(texture) glTexCoord2d(1, texT);
      v = vl[0];
      glVertex3d(v[0], y, v[1]);
```

```
      prevTexT = texT;
      prevY = y;

      glEnd();
    }

    if(showNormals) for(int i=0; i<thetaSteps; ++i)
    {
      const double texS = 1.0 / thetaSteps * i;
      const double theta = 2 * M_PI * texS;

      double x = -sin(theta);
      double z = -cos(theta);

      normals.push_back(Vector3(vl[i][0], prevY, vl[i][1]));
      normals.push_back(Vector3(x*n[0], n[1], z*n[0]));
    }
  }

  if(showNormals) renderNormals(normals, config);
}

void Cylinder::render(bool texture, bool color,
                      const Shape *shape,
                      const RenderConfig *config) const
{
  glFrontFace(GL_CCW);
  glDisable(GL_COLOR_MATERIAL);
  if(side&top&bottom)
  {
    glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, 0);
    glEnable(GL_CULL_FACE);
  }
  else
  {
    glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, 1);
    glDisable(GL_CULL_FACE);
  }

  const int thetaSteps = config->subdivision1; // Azimuth / longitude

  const double r = radius;
  const double y = height / 2;

  vector<Vector2> vl(thetaSteps);
  vector<Vector2> tl(texture ? thetaSteps : 0);

  vector<Vector3> normals;
  const bool showNormals = config->showNormals;

  // Side
  if(side)
  {
    glBegin(GL_QUAD_STRIP);

    for(int i=0; i<thetaSteps; ++i)
    {
      const double texS = 1.0 / thetaSteps * i;
```



```
      const double theta = 2 * M_PI * texS;

      double x = -sin(theta);
      double z = -cos(theta);

      if(texture) tl[i] = Vector2(x/2+0.5, z/-2+0.5);

      glNormal3d(x, 0, z);
      if(showNormals)
      {
        normals.push_back(Vector3(x*r, y, z*r));
        normals.push_back(Vector3(x, 0, z));
        normals.push_back(Vector3(x*r, -y, z*r));
        normals.push_back(Vector3(x, 0, z));
      }
      x *= r;
      z *= r;
      if(texture) glTexCoord2d(texS, 1);
      glVertex3d(x, y, z);
      if(texture) glTexCoord2d(texS, 0);
      glVertex3d(x, -y, z);

      vl[i] = Vector2(x, z);
    }

    glNormal3d(0, 0, -1);
    if(texture) glTexCoord2d(1, 1);
    glVertex3d(0, y, -r);
    if(texture) glTexCoord2d(1, 0);
    glVertex3d(0, -y, -r);

    glEnd();
  }
  else
  {
    for(int i=0; i<thetaSteps; ++i)
    {
      const double theta = 2 * M_PI / thetaSteps * i;

      double x = -sin(theta);
      double z = -cos(theta);
      if(texture) tl[i] = Vector2(x/2+0.5, z/-2+0.5);
      vl[i] = Vector2(x*r, z*r);
    }
  }

  // Top
  if(top)
  {
    glNormal3d(0, 1, 0);
    glBegin(GL_POLYGON);
    for(int i=0; i<thetaSteps; ++i)
    {
      if(texture) glTexCoord2d(tl[i][0], tl[i][1]);
      glVertex3d(vl[i][0], y, vl[i][1]);
    }
    glEnd();
    if(showNormals) for(int i=0; i<thetaSteps; ++i)
```

```
      {
        normals.push_back(Vector3(vl[i][0], y, vl[i][1]));
        normals.push_back(Vector3(0, 1, 0));
      }
    }

    // Bottom
    if(bottom)
    {
      glNormal3d(0, -1, 0);
      glBegin(GL_POLYGON);
      for(int i=thetaSteps-1; i>=0; --i)
      {
        if(texture) glTexCoord2d(tl[i][0], -tl[i][1]);
        glVertex3d(vl[i][0], -y, vl[i][1]);
      }
      glEnd();
      if(showNormals) for(int i=0; i<thetaSteps; ++i)
      {
        normals.push_back(Vector3(vl[i][0], -y, vl[i][1]));
        normals.push_back(Vector3(0, -1, 0));
      }
    }

    if(showNormals) renderNormals(normals, config);
  }


  void IndexedLineSet::render(bool texture, bool allowColor,
                              const Shape *shape,
                              const RenderConfig *config) const
  {
    if(!coord) return;

    const Coordinate *_coord =
      dynamic_cast<const Coordinate *>(coord.get());
    const Color *rgb =
      dynamic_cast<const Color *>(color.get());
    const ColorRGBA *rgba =
      dynamic_cast<const ColorRGBA *>(color.get());

    if(!_coord)
      ARCHON_THROW1(InternalException,
                    "'" + coord->getType()->getName() +
                    "' is not supported yet");
    if(!rgb && !rgba && color)
      ARCHON_THROW1(InternalException,
                    "'" + color->getType()->getName() +
                    "' is not supported yet");

    const vector<Vector3> &v = _coord->getPoint();
    const vector<Vector3> *rgb_v  = rgb  ? &rgb->getColor()  : 0;
    const vector<Vector4> *rgba_v = rgba ? &rgba->getColor() : 0;

    bool hasColorIndex = !colorIndex.empty();
    bool rgbPerVertex  = rgb  && colorPerVertex;
    bool rgbPerFace    = rgb  && !rgbPerVertex;
    bool rgbaPerVertex = rgba && colorPerVertex;
```



```
    bool rgbaPerFace   = rgba && !rgbaPerVertex;

    vector<int>::const_iterator ci = colorIndex.begin();
    vector<int>::const_iterator vi = coordIndex.begin();

    glLineWidth(lineWidth);
    //glEnable(GL_LINE_SMOOTH);
    glDisable(GL_LIGHTING);
    glDisable(GL_TEXTURE_2D);

    if(!rgb && !rgba)
    {
      const Appearance *a =
        dynamic_cast<const Appearance *>(shape->getAppearance().get());
      if(a)
      {
        const Material *m =
          dynamic_cast<const Material *>(a->getMaterial().get());
        if(m)
        {
          Vector3 c = m->getEmissiveColor();
          glColor3d(c[0], c[1], c[2]);
        }
      }
    }

    int i = 0; // Face index
    while(vi != coordIndex.end())
    {
      if(rgbPerFace)
      {
        const Vector3 &c = (*rgb_v)[hasColorIndex ? *(ci++) : i];
        glColor3d(c[0], c[1], c[2]);
      }

      if(rgbaPerFace)
      {
        const Vector4 &c = (*rgba_v)[hasColorIndex ? *(ci++) : i];
        glColor4d(c[0], c[1], c[2], c[3]);
      }

      glBegin(GL_LINE_STRIP);
      for(;;)
      {
        if(rgbPerVertex)
        {
          const Vector3 &c = (*rgb_v)[hasColorIndex ? *(ci++) : *vi];
          glColor3d(c[0], c[1], c[2]);
        }

        if(rgbaPerVertex)
        {
          const Vector4 &c = (*rgba_v)[hasColorIndex ? *(ci++) : *vi];
          glColor4d(c[0], c[1], c[2], c[3]);
        }

        const Vector3 &w = v[*(vi++)];
        glVertex3d(w[0], w[1], w[2]);
```

```
      if(vi == coordIndex.end()) break;
      if(*vi == -1)
      {
        ++vi;
        if(rgbPerVertex || rgbaPerVertex) ++ci;
        break;
      }
    }
    glEnd();

    ++i;
  }
}


void PointSet::render(bool texture, bool allowColor,
                      const Shape *shape,
                      const RenderConfig *config) const
{
  if(!coord) return;

  const Coordinate *_coord =
    dynamic_cast<const Coordinate *>(coord.get());
  const Color *rgb =
    dynamic_cast<const Color *>(color.get());
  const ColorRGBA *rgba =
    dynamic_cast<const ColorRGBA *>(color.get());

  if(!_coord)
    ARCHON_THROW1(InternalException,
                  "'" + coord->getType()->getName() +
                  "' is not supported yet");
  if(!rgb && !rgba && color)
    ARCHON_THROW1(InternalException,
                  "'" + color->getType()->getName() +
                  "' is not supported yet");

  const vector<Vector3> &v = _coord->getPoint();
  const unsigned n = v.size();
  if(!n) return;

  if(rgb  && rgb->getColor().size()  < n ||
     rgba && rgba->getColor().size() < n ||
     !allowColor)
  {
    rgb = 0;
    rgba = 0;
  }

  glPointSize(2);
  //glEnable(GL_POINT_SMOOTH);
  glDisable(GL_LIGHTING);
  glDisable(GL_TEXTURE_2D);

  if(!rgb && !rgba)
  {
    const Appearance *a =
```



```
      dynamic_cast<const Appearance *>(shape->getAppearance().get());
    if(a)
    {
      const Material *m =
        dynamic_cast<const Material *>(a->getMaterial().get());
      if(m)
      {
        Vector3 c = m->getEmissiveColor();
        glColor3d(c[0], c[1], c[2]);
      }
    }
  }

  glBegin(GL_POINTS);
  for(unsigned i=0; i<n; ++i)
  {
    if(rgb)
    {
      Vector3 c = rgb->getColor()[i];
      glColor3d(c[0], c[1], c[2]);
    }
    else if(rgba)
    {
      Vector4 c = rgba->getColor()[i];
      glColor4d(c[0], c[1], c[2], c[3]);
    }
    Vector3 w = v[i];
    glVertex3d(w[0], w[1], w[2]);
  }
  glEnd();
}


void Sphere::render(bool texture, bool color,
                    const Shape *shape,
                    const RenderConfig *config) const
{
  glFrontFace(GL_CW);
  glDisable(GL_COLOR_MATERIAL);
  glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, 0);
  glEnable(GL_CULL_FACE);

  const int phiSteps = config->subdivision2;  // Eleveation / lattitude
  const int thetaSteps = config->subdivision1; // Azimuth / longitude

  vector<Vector3> normals;
  const bool showNormals = config->showNormals;

  vector<Vector3> vl(thetaSteps);

  glBegin(GL_QUAD_STRIP);

  Vector3 v;

  // Southern pole
  if(showNormals)
  {
    normals.push_back(Vector3(0, -radius, 0));
```

```
  normals.push_back(Vector3(0, -1, 0));
}
double texT = 1.0 / phiSteps;
double phi = M_PI * (texT - 0.5);
double sin_phi = sin(phi);
double cos_phi = cos(phi);
for(int i=0; i<thetaSteps; ++i)
{
  double texS = 1.0 / thetaSteps * i;
  double theta = 2 * M_PI * texS;

  if(texture) glTexCoord2d(texS, 0);
  glNormal3d(0, -1, 0);
  glVertex3d(0, -radius, 0);

  if(texture) glTexCoord2d(texS, texT);
  v = vl[i] = Vector3(cos_phi*-sin(theta), sin_phi, cos_phi*-cos(theta
       ));
  glNormal3d(v[0], v[1], v[2]);
  v *= radius;
  glVertex3d(v[0], v[1], v[2]);
  if(showNormals)
  {
    normals.push_back(v);
    normals.push_back(vl[i]);
  }
}

if(texture) glTexCoord2d(1, 0);
glNormal3d(0, -1, 0);
glVertex3d(0, -radius, 0);

if(texture) glTexCoord2d(1, texT);
v = vl[0];
glNormal3d(v[0], v[1], v[2]);
v *= radius;
glVertex3d(v[0], v[1], v[2]);

double prevTexT = texT;

glEnd();

// Middle quad strips
for(int i=2; i<phiSteps; ++i)
{
  glBegin(GL_QUAD_STRIP);

  Vector3 w = vl[0];

  texT = 1.0 / phiSteps * i;
  phi = M_PI * (texT - 0.5);
  sin_phi = sin(phi);
  cos_phi = cos(phi);
  for(int j=0; j<thetaSteps; ++j)
  {
    double texS = 1.0 / thetaSteps * j;
    double theta = 2 * M_PI * texS;
```



```
    if(texture) glTexCoord2d(texS, prevTexT);
    v = vl[j];
    glNormal3d(v[0], v[1], v[2]);
    v *= radius;
    glVertex3d(v[0], v[1], v[2]);

    if(texture) glTexCoord2d(texS, texT);
    v = vl[j] = Vector3(cos_phi*-sin(theta), sin_phi, cos_phi*-cos(
         theta));
    glNormal3d(v[0], v[1], v[2]);
    v *= radius;
    glVertex3d(v[0], v[1], v[2]);
    if(showNormals)
    {
      normals.push_back(v);
      normals.push_back(vl[j]);
    }
  }

  if(texture) glTexCoord2d(1, prevTexT);
  glNormal3d(w[0], w[1], w[2]);
  w *= radius;
  glVertex3d(w[0], w[1], w[2]);

  if(texture) glTexCoord2d(1, texT);
  v = vl[0];
  glNormal3d(v[0], v[1], v[2]);
  v *= radius;
  glVertex3d(v[0], v[1], v[2]);

  prevTexT = texT;

  glEnd();
}

// Northern pole
glBegin(GL_QUAD_STRIP);

for(int i=0; i<thetaSteps; ++i)
{
  double texS = 1.0 / thetaSteps * i;

  if(texture) glTexCoord2d(texS, prevTexT);
  v = vl[i];
  glNormal3d(v[0], v[1], v[2]);
  v *= radius;
  glVertex3d(v[0], v[1], v[2]);

  if(texture) glTexCoord2d(texS, 1);
  glNormal3d(0, 1, 0);
  glVertex3d(0, radius, 0);
}

if(texture) glTexCoord2d(1, prevTexT);
v = vl[0];
glNormal3d(v[0], v[1], v[2]);
v *= radius;
glVertex3d(v[0], v[1], v[2]);
```

```
  if(texture) glTexCoord2d(1, 1);
  glNormal3d(0, 1, 0);
  glVertex3d(0, radius, 0);

  if(showNormals)
  {
    normals.push_back(Vector3(0, radius, 0));
    normals.push_back(Vector3(0, 1, 0));
  }

  glEnd();

  if(showNormals) renderNormals(normals, config);
}


/*
void concavePolygonDecompose(const vector<Vector3> &vertices,
                             const vector<int>)
{

}
*/


/*
 * Crease-angle:
 *
 * Regardless to the value of 'normalPerVertex', if the 'normal'
 * field of the indexed face set is null, then normals must be
 * generated with respect to the value of 'creaseAngle'.
 *
 * We do this by first calculating a "physical" normal for each
 * face using the cross product. At the same time we build up
 * lists of faces for each vertex.
 *
 * Preparing these two sets of information will speed up the
 * overall process. Typically a list of faces for a specific
 * vertex will be needed 3-4 times, and the normal for a face will
 * be needed 12-16 times, depending on wether we are deling with a
 * mesh of triangles or it is a mesh of quadrilaterals.
 *
 * Now, when we render face F, we will, for each vertex, compute
 * the effective normal as the average between all the normals of
 * the qualifying faces of the face list associated with the
 * current vertex. A face is qualifying if the angle between its
 * normal and F is less than 'creaseAngle'.
 *
 * Note: By doing it this way we interpret the phrase "two
 * adjacent polygonal faces" from the X3D specification (Part 1 -
 * 11.2.3 Common geometry fields) as "having at least one vertex
 * in common". The alternative interpretation would be "having at
 * least one edge in common" but this would both be harder to
 * implement (do intersection between face lists) and it would not
 * work properly in the very simple situation where four
 * quadrilaterals share a vertex. Two of them will not have any
 * edges in common and thus use different normals in the shared
```



```
 * vertex.
 *
 * \todo Vulnerability to invalid indices + less than three verts
 * per poly. May cause SegFault.
 *
 * \todo Polygons whos first 3 vertices are co-linear (lie on the
 * same straight line) will cheat the automatic normal generation.
 *
 * \todo Support non-convex polygons as faces. One way to deal
 * with it would be to walk around the polygon and recursively
 * restart at every vertex where the angle is numb. A specific
 * recursion stops before the first vertex that passes the line
 * coincident with the first edge of this recursion.
 *
 * \todo Talk to Andreas about deploying the angle based weighd
 * avaraging algorithm.
 *
 * \todo Verify that the matter discussed in the 'note' is
 * implementd as required by the X3D specification.
 *
 * \todo Skip all the heavy creaseAngle stuff if it is set to
 * zero. Also it may be optimized if creaseAngle>=2*pi.
 */
void IndexedFaceSet::render(bool texture, bool allowColor,
                            const Shape *shape,
                            const RenderConfig *config) const
{
  if(!coord) return;

  const Coordinate *_coord =
    dynamic_cast<const Coordinate *>(coord.get());
  const Color *rgb =
    dynamic_cast<const Color *>(color.get());
  const ColorRGBA *rgba =
    dynamic_cast<const ColorRGBA *>(color.get());
  const TextureCoordinate *_texCoord =
    dynamic_cast<const TextureCoordinate *>(texCoord.get());
  const Normal *_normal =
    dynamic_cast<const Normal *>(normal.get());

  if(!_coord)
    ARCHON_THROW1(InternalException,
                  "'" + coord->getType()->getName() +
                  "' is not supported yet");
  if(!rgb && !rgba && color)
    ARCHON_THROW1(InternalException,
                  "'" + color->getType()->getName() +
                  "' is not supported yet");
  if(!_texCoord && texCoord)
    ARCHON_THROW1(InternalException,
                  "'" + texCoord->getType()->getName() +
                  "' is not supported yet");
  if(!_normal && normal)
    ARCHON_THROW1(InternalException,
                  "'" + normal->getType()->getName() +
                  "' is not supported yet");

  if(!texture) _texCoord = 0;
```

```
if(!allowColor)
{
  rgb = 0;
  rgba = 0;
}

bool hasColorIndex    = !colorIndex.empty();
bool hasTexCoordIndex = !texCoordIndex.empty();
bool hasNormalIndex   = !normalIndex.empty();

bool rgbPerVertex     = rgb     && colorPerVertex;
bool rgbPerFace       = rgb     && !rgbPerVertex;
bool rgbaPerVertex    = rgba    && colorPerVertex;
bool rgbaPerFace      = rgba    && !rgbaPerVertex;
bool _normalPerVertex = _normal && normalPerVertex;
bool normalPerFace    = _normal && !_normalPerVertex;

glFrontFace(ccw ? GL_CCW : GL_CW);

if(solid)
{
  glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, 0);
  glEnable(GL_CULL_FACE);
}
else
{
  glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, 1);
  glDisable(GL_CULL_FACE);
}

if(colorPerVertex || rgbPerFace || rgbaPerFace)
  glEnable(GL_COLOR_MATERIAL);
else
  glDisable(GL_COLOR_MATERIAL);

vector<int>::const_iterator ci = colorIndex.begin();
vector<int>::const_iterator ti = texCoordIndex.begin();
vector<int>::const_iterator ni = normalIndex.begin();
vector<int>::const_iterator vi = coordIndex.begin();

const vector<Vector3> *rgb_v  = rgb       ? &rgb->getColor()       : 0;
const vector<Vector4> *rgba_v = rgba      ? &rgba->getColor()      : 0;
const vector<Vector2> *tv     = _texCoord ? &_texCoord->getPoint() : 0;
const vector<Vector3> *nv     = _normal   ? &_normal->getNormal()  : 0;
const vector<Vector3> *vv     =             &_coord->getPoint();

if(texture && !_texCoord)
{
  // Setup automatic texture coordinate generation
  Vector3 bboxCenter = shape->getBboxCenter();
  Vector3 bboxSize = shape->getBboxSize();
  if(bboxSize[0] == -1)
  {
    // Compute bounding box
    Box3 b(Vector3::zero());
    for(unsigned i=0; i<vv->size(); ++i)
    {
      Vector3 v = (*vv)[i];
```



```
      if(v[0] < b.lowerCorner[0]) b.lowerCorner[0] = v[0];
      if(v[0] > b.upperCorner[0]) b.upperCorner[0] = v[0];
      if(v[1] < b.lowerCorner[1]) b.lowerCorner[1] = v[1];
      if(v[1] > b.upperCorner[1]) b.upperCorner[1] = v[1];
      if(v[2] < b.lowerCorner[2]) b.lowerCorner[2] = v[2];
      if(v[2] > b.upperCorner[2]) b.upperCorner[2] = v[2];
    }
    bboxCenter = b.getCenter();
    bboxSize = b.getSize();
  }

  // Find out which spatial coordinates map to which
  // texture coordinates
  vector<pair<double, int> > m;
  m.push_back(make_pair(bboxSize[0], 0));
  vector<pair<double, int> >::iterator i = m.begin();
  while(i->first >= bboxSize[1]) ++i;
  m.insert(i, make_pair(bboxSize[1], 1));
  i = m.begin();
  while(i->first >= bboxSize[2]) ++i;
  m.insert(i, make_pair(bboxSize[2], 2));
  i = m.begin();

  // First texture coordinate S
  {
    double c = 1/bboxSize[i->second];
    GLdouble v[4] = { 0, 0, 0, (bboxSize[i->second]/2-bboxCenter[i->
         second])*c };
    v[i->second] = c;
    glTexGendv(GL_S, GL_OBJECT_PLANE, v);
  }

  ++i;

  // Second texture coordinate T
  {
    double c = 1/bboxSize[i->second];
    GLdouble v[4] = { 0, 0, 0, (bboxSize[i->second]/2-bboxCenter[i->
         second])*c };
    v[i->second] = c;
    glTexGendv(GL_T, GL_OBJECT_PLANE, v);
  }

  glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
  glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);

  glEnable(GL_TEXTURE_GEN_S);
  glEnable(GL_TEXTURE_GEN_T);
}

int i = 0; // Face index
int j = 0; // Vertex index

/*
 * Preperation for automatic normal computation
 *
 * Note: It may be more efficient to start determining the
 * number of faces and then resize the autoNormal vector
```

```
   * initially than progressivly resizeing it.
   */
  double creaseDotLimit; // Limit on normal dot-product alt. to limit on
       angle
  vector<Vector3> autoNormals;
  vector<vector<int> > vertexFaces;
  if(!_normal)
  {
    creaseDotLimit = creaseAngle >= M_PI ? -1 : cos(creaseAngle);
    vertexFaces.resize(_coord->getPoint().size());
    const bool cw = !ccw;
    while(vi != coordIndex.end())
    {
      Vector3 n = (*vv)[*(vi+1)];
      n -= (*vv)[*vi];
      Vector3 v = (*vv)[*(vi+2)];
      v -= (*vv)[*(vi+1)];
      n *= v;
      if(cw) n.negate();
      n.normalize();
      autoNormals.push_back(n);
      while(vi != coordIndex.end() && *vi != -1)
      {
        vertexFaces[*vi].push_back(i);
        ++vi;
      }
      if(vi != coordIndex.end()) ++vi;
      ++i;
    }

    i = 0;
    vi = coordIndex.begin();
  }

  vector<Vector3> normals;
  const bool showNormals = config->showNormals;

  while(vi != coordIndex.end())
  {
    if(rgbPerFace)
    {
      const Vector3 &c = (*rgb_v)[hasColorIndex ? *(ci++) : i];
      glColor3d(c[0], c[1], c[2]);
    }

    if(rgbaPerFace)
    {
      const Vector4 &c = (*rgba_v)[hasColorIndex ? *(ci++) : i];
      glColor4d(c[0], c[1], c[2], c[3]);
    }

    Vector3 n;

    if(normalPerFace)
    {
      n = (*nv)[hasNormalIndex ? *(ni++) : i];
      glNormal3d(n[0], n[1], n[2]);
    }
```



```
    glBegin(GL_POLYGON);
    for(;;)
    {
      if(rgbPerVertex)
      {
        const Vector3 &c = (*rgb_v)[hasColorIndex ? *(ci++) : *vi];
        glColor3d(c[0], c[1], c[2]);
      }

      if(rgbaPerVertex)
      {
        const Vector4 &c = (*rgba_v)[hasColorIndex ? *(ci++) : *vi];
        glColor4d(c[0], c[1], c[2], c[3]);
      }

      if(tv)
      {
        const Vector2 &t = (*tv)[hasTexCoordIndex ? *(ti++) : *vi];
        glTexCoord2d(t[0], t[1]);
      }

      if(_normalPerVertex)
      {
        n = (*nv)[hasNormalIndex ? *(ni++) : *vi];
        glNormal3d(n[0], n[1], n[2]);
      }
      else if(!normalPerFace)
      {
        // Auto generate normal

        Vector3 n0 = autoNormals[i];
        vector<int> &fv = vertexFaces[*vi];
        vector<int>::iterator fi = fv.begin();

        n = n0; // Sum of normals
        int k = 1; // Number of added normals;

        while(fi != fv.end())
        {
          if(*fi != i)
          {
            Vector3 n1 = autoNormals[*fi];

            if(dot(n0, n1) > creaseDotLimit)
            {
              n += n1;
              ++k;
            }
          }

          ++fi;
        }

        n /= k;

        glNormal3d(n[0], n[1], n[2]);
      }
```

```
        const Vector3 &v = (*vv)[*(vi++)];
        glVertex3d(v[0], v[1], v[2]);
        if(showNormals)
        {
          normals.push_back(v);
          normals.push_back(n);
        }

        if(vi == coordIndex.end()) break;
        if(*vi == -1)
        {
          ++vi;
          if(rgbPerVertex || rgbaPerVertex) ++ci;
          if(tv) ++ti;
          if(_normalPerVertex) ++ni;
          break;
        }

        ++j;
      }
      glEnd();

      ++i;
    }

    if(texture && !_texCoord)
    {
      glDisable(GL_TEXTURE_GEN_S);
      glDisable(GL_TEXTURE_GEN_T);
    }

    if(showNormals) renderNormals(normals, config);
  }

  /**
   * \todo Considder using glMapGrid and glEvalMesh to ease this
   * job.
   */
  void ElevationGrid::render(bool texture, bool allowColor,
                             const Shape *shape,
                             const RenderConfig *config) const
  {
    const unsigned n = xDimension * zDimension;
    if(height.size() < n) return;

    const int iMax = xDimension - 1;
    const int jMax = zDimension - 1;
    const unsigned m = iMax * jMax;

    const Color *rgb =
      dynamic_cast<const Color *>(color.get());
    const ColorRGBA *rgba =
      dynamic_cast<const ColorRGBA *>(color.get());
    const TextureCoordinate *_texCoord =
      dynamic_cast<const TextureCoordinate *>(texCoord.get());
    const Normal *_normal =
      dynamic_cast<const Normal *>(normal.get());
```



```
    if(!rgb && !rgba && color)
      ARCHON_THROW1(InternalException,
                    "'" + color->getType()->getName() +
                    "' is not supported yet");
    if(!_texCoord && texCoord)
      ARCHON_THROW1(InternalException,
                    "'" + texCoord->getType()->getName() +
                    "' is not supported yet");
    if(!_normal && normal)
      ARCHON_THROW1(InternalException,
                    "'" + normal->getType()->getName() +
                    "' is not supported yet");

    if(rgb  && rgb->getColor().size()  < (colorPerVertex ? n : m) ||
       rgba && rgba->getColor().size() < (colorPerVertex ? n : m) ||
       !allowColor)
    {
      rgb = 0;
      rgba = 0;
    }

    if(_texCoord && _texCoord->getPoint().size() < n) _texCoord = 0;
    if(_normal && _normal->getNormal().size() < (normalPerVertex ? n : m))
        _normal = 0;

    glFrontFace(ccw ? GL_CCW : GL_CW);

    if(solid)
    {
      glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, 0);
      glEnable(GL_CULL_FACE);
    }
    else
    {
      glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, 1);
      glDisable(GL_CULL_FACE);
    }

    if(rgb || rgba) glEnable(GL_COLOR_MATERIAL);
    else glDisable(GL_COLOR_MATERIAL);


    /*
     * Preperation for automatic normal computation
     */
    double creaseDotLimit; // Limit on normal dot-product alt. to limit on
        angle
    vector<Vector3> autoNormals(m);
    double prevZ = 0;
    if(!_normal)
    {
      creaseDotLimit = creaseAngle >= M_PI ? -1 : cos(creaseAngle);
      const bool cw = !ccw;
      for(int j=1; j<zDimension; ++j)
      {
        const double z = zSpacing * j;
```

```
    double prevX = 0;
    for(int i=1; i<xDimension; ++i)
    {
      const double x = xSpacing * i;

      Vector3 n(prevX, height[(i-1) + (j-1) * xDimension], prevZ);
      n -= Vector3(x, height[i + j * xDimension], z);
      Vector3 v(prevX, height[(i-1) + j * xDimension], z);
      v -= Vector3(x, height[i + (j-1) * xDimension], prevZ);
      n *= v;
      if(cw) n.negate();
      n.normalize();
      autoNormals[(i-1) + (j-1) * iMax] = n;

      prevX = x;
    }

    prevZ = z;
  }

  prevZ = 0;
}

double prevT = 0;

vector<Vector3> normals;
const bool showNormals = config->showNormals;

glBegin(GL_QUADS);

for(int j=1; j<zDimension; ++j)
{
  const double z = zSpacing * j;
  const double t = j/double(jMax);

  double prevX = 0;
  double prevS = 0;

  for(int i=1; i<xDimension; ++i)
  {
    const double x = xSpacing * i;
    const double s = i/double(iMax);

    int faseIndex = (i-1) + (j-1) * iMax;

    Vector3 n;

    if(rgb && !colorPerVertex)
    {
      const Vector3 &c = rgb->getColor()[faseIndex];
      glColor3d(c[0], c[1], c[2]);
    }
    if(rgba && !colorPerVertex)
    {
      const Vector4 &c = rgba->getColor()[faseIndex];
      glColor4d(c[0], c[1], c[2], c[3]);
    }
    if(_normal && !normalPerVertex)
```



```
    {
      n = _normal->getNormal()[faseIndex];
      glNormal3d(n[0], n[1], n[2]);
    }


    // First vertex (NE)
    {
      int vertexIndex = i + (j-1) * xDimension;

      if(rgb && colorPerVertex)
      {
        const Vector3 &c = rgb->getColor()[vertexIndex];
        glColor3d(c[0], c[1], c[2]);
      }
      if(rgba && colorPerVertex)
      {
        const Vector4 &c = rgba->getColor()[vertexIndex];
        glColor4d(c[0], c[1], c[2], c[3]);
      }
      if(_normal && normalPerVertex)
      {
        n = _normal->getNormal()[vertexIndex];
        glNormal3d(n[0], n[1], n[2]);
      }
      else if(!_normal)
      {
        // Auto generate normal

        Vector3 n0 = autoNormals[faseIndex];
        n = n0;
        int m = 1;
        if(i<iMax)
        {
          Vector3 n1 = autoNormals[faseIndex + 1];
          if(dot(n0, n1) > creaseDotLimit) { n += n1; ++m; }
        }
        if(j>1)
        {
          Vector3 n1 = autoNormals[faseIndex - iMax];
          if(dot(n0, n1) > creaseDotLimit) { n += n1; ++m; }
        }
        if(i<iMax && j>1)
        {
          Vector3 n1 = autoNormals[faseIndex - iMax + 1];
          if(dot(n0, n1) > creaseDotLimit) { n += n1; ++m; }
        }
        n /= m;

        glNormal3d(n[0], n[1], n[2]);
      }

      if(texture)
      {
        if(texCoord)
        {
          const Vector2 &v = _texCoord->getPoint()[vertexIndex];
          glTexCoord2d(v[0], v[1]);
```

```
      }
      else glTexCoord2d(s, prevT);
    }

    glVertex3d(x, height[vertexIndex], prevZ);
    if(showNormals)
    {
      normals.push_back(Vector3(x, height[vertexIndex], prevZ));
      normals.push_back(n);
    }
  }

  // Second vertex (NW)
  {
    int vertexIndex = (i-1) + (j-1) * xDimension;

    if(rgb && colorPerVertex)
    {
      const Vector3 &c = rgb->getColor()[vertexIndex];
      glColor3d(c[0], c[1], c[2]);
    }
    if(rgba && colorPerVertex)
    {
      const Vector4 &c = rgba->getColor()[vertexIndex];
      glColor4d(c[0], c[1], c[2], c[3]);
    }
    if(_normal && normalPerVertex)
    {
      n = _normal->getNormal()[vertexIndex];
      glNormal3d(n[0], n[1], n[2]);
    }
    else if(!_normal)
    {
      // Auto generate normal

      Vector3 n0 = autoNormals[faseIndex];
      n = n0;
      int m = 1;
      if(i>1)
      {
        Vector3 n1 = autoNormals[faseIndex - 1];
        if(dot(n0, n1) > creaseDotLimit) { n += n1; ++m; }
      }
      if(j>1)
      {
        Vector3 n1 = autoNormals[faseIndex - iMax];
        if(dot(n0, n1) > creaseDotLimit) { n += n1; ++m; }
      }
      if(i>1 && j>1)
      {
        Vector3 n1 = autoNormals[faseIndex - iMax - 1];
        if(dot(n0, n1) > creaseDotLimit) { n += n1; ++m; }
      }
      n /= m;

      glNormal3d(n[0], n[1], n[2]);
    }
```



```
    if(texture)
    {
      if(texCoord)
      {
        const Vector2 &v = _texCoord->getPoint()[vertexIndex];
        glTexCoord2d(v[0], v[1]);
      }
      else glTexCoord2d(prevS, prevT);
    }

    glVertex3d(prevX, height[vertexIndex], prevZ);
    if(showNormals)
    {
      normals.push_back(Vector3(prevX, height[vertexIndex], prevZ));
      normals.push_back(n);
    }
  }

  // Third vertex (SW)
  {
    int vertexIndex = (i-1) + j * xDimension;

    if(rgb && colorPerVertex)
    {
      const Vector3 &c = rgb->getColor()[vertexIndex];
      glColor3d(c[0], c[1], c[2]);
    }
    if(rgba && colorPerVertex)
    {
      const Vector4 &c = rgba->getColor()[vertexIndex];
      glColor4d(c[0], c[1], c[2], c[3]);
    }
    if(_normal && normalPerVertex)
    {
      n = _normal->getNormal()[vertexIndex];
      glNormal3d(n[0], n[1], n[2]);
    }
    else if(!_normal)
    {
      // Auto generate normal

      Vector3 n0 = autoNormals[faseIndex];
      n = n0;
      int m = 1;
      if(i>1)
      {
        Vector3 n1 = autoNormals[faseIndex - 1];
        if(dot(n0, n1) > creaseDotLimit) { n += n1; ++m; }
      }
      if(j<jMax)
      {
        Vector3 n1 = autoNormals[faseIndex + iMax];
        if(dot(n0, n1) > creaseDotLimit) { n += n1; ++m; }
      }
      if(i>1 && j<jMax)
      {
        Vector3 n1 = autoNormals[faseIndex + iMax - 1];
        if(dot(n0, n1) > creaseDotLimit) { n += n1; ++m; }
```

```
      }
      n /= m;

      glNormal3d(n[0], n[1], n[2]);
    }

    if(texture)
    {
      if(texCoord)
      {
        const Vector2 &v = _texCoord->getPoint()[vertexIndex];
        glTexCoord2d(v[0], v[1]);
      }
      else glTexCoord2d(prevS, t);
    }

    glVertex3d(prevX, height[vertexIndex], z);
    if(showNormals)
    {
      normals.push_back(Vector3(prevX, height[vertexIndex], z));
      normals.push_back(n);
    }
  }

  // Fourth vertex (SE)
  {
    int vertexIndex = i + j * xDimension;

    if(rgb && colorPerVertex)
    {
      const Vector3 &c = rgb->getColor()[vertexIndex];
      glColor3d(c[0], c[1], c[2]);
    }
    if(rgba && colorPerVertex)
    {
      const Vector4 &c = rgba->getColor()[vertexIndex];
      glColor4d(c[0], c[1], c[2], c[3]);
    }
    if(_normal && normalPerVertex)
    {
      n = _normal->getNormal()[vertexIndex];
      glNormal3d(n[0], n[1], n[2]);
    }
    else if(!_normal)
    {
      // Auto generate normal

      Vector3 n0 = autoNormals[faseIndex];
      n = n0;
      int m = 1;
      if(i<iMax)
      {
        Vector3 n1 = autoNormals[faseIndex + 1];
        if(dot(n0, n1) > creaseDotLimit) { n += n1; ++m; }
      }
      if(j<jMax)
      {
        Vector3 n1 = autoNormals[faseIndex + iMax];
```



```
        if(dot(n0, n1) > creaseDotLimit) { n += n1; ++m; }
      }
      if(i<iMax && j<jMax)
      {
        Vector3 n1 = autoNormals[faseIndex + iMax + 1];
        if(dot(n0, n1) > creaseDotLimit) { n += n1; ++m; }
      }
      n /= m;

      glNormal3d(n[0], n[1], n[2]);
    }

    if(texture)
    {
      if(texCoord)
      {
        const Vector2 &v = _texCoord->getPoint()[vertexIndex];
        glTexCoord2d(v[0], v[1]);
      }
      else glTexCoord2d(s, t);
    }

    glVertex3d(x, height[vertexIndex], z);
    if(showNormals)
    {
      normals.push_back(Vector3(x, height[vertexIndex], z));
      normals.push_back(n);
    }
  }

  prevX = x;
  prevS = s;
  }

  prevZ = z;
  prevT = t;
  }

  glEnd();

  if(showNormals) renderNormals(normals, config);
  }
  }
}
```

Listing: **archon/x3d/server/geoprop.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
```

```
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_GEOPROP_H
#define ARCHON_X3D_GEOPROP_H

/**
 * \file geoprop.H
 *
 * X3D Geometric properties component
 */

#include <vector>

#include <archon/math/vector.H>

#include <archon/x3d/server/core.H>
#include <archon/x3d/server/scene.H>

namespace Archon
{
  namespace X3D
  {
    using namespace Math;


    struct GeometricPropertyNode: virtual Node
    {
      static const NodeType *type;

    protected:
      GeometricPropertyNode() {}
    };


    struct ColorNode: GeometricPropertyNode
    {
      static const NodeType *type;

    protected:
      ColorNode() {}
    };


    struct CoordinateNode: GeometricPropertyNode
    {
      static const NodeType *type;

    protected:
      CoordinateNode() {}
    };
```



```
    struct NormalNode: GeometricPropertyNode
    {
      static const NodeType *type;

    protected:
      NormalNode() {}
    };


    struct Color: ColorNode
    {
      static Ref<Node> instantiate(BackRef<ExecutionContext> c)
      {
        return new Color(c);
      }

      static const NodeType *type;
      const NodeType *getType() const { return type; }

      Color(BackRef<ExecutionContext> c): Node(c) {}

      const vector<Vector3> &getColor() const { return color; }

    private:
      friend void initializeGeopropComponent();

      vector<Vector3>  color;

      EventSource colorChanged;

      Time colorStamp;
    };



    struct ColorRGBA: ColorNode
    {
      static Ref<Node> instantiate(BackRef<ExecutionContext> c)
      {
        return new ColorRGBA(c);
      }

      static const NodeType *type;
      const NodeType *getType() const { return type; }

      ColorRGBA(BackRef<ExecutionContext> c): Node(c) {}

      const vector<Vector4> &getColor() const { return color; }

    private:
      friend void initializeGeopropComponent();

      vector<Vector4>  color;

      EventSource colorChanged;

      Time colorStamp;
```

```
    };


    struct Coordinate: CoordinateNode
    {
      static Ref<Node> instantiate(BackRef<ExecutionContext> c)
      {
        return new Coordinate(c);
      }

      static const NodeType *type;
      const NodeType *getType() const { return type; }

      Coordinate(BackRef<ExecutionContext> c): Node(c) {}

      const vector<Vector3> &getPoint() const { return point; }

    private:
      friend void initializeGeopropComponent();

      vector<Vector3>  point;

      EventSource pointChanged;

      Time pointStamp;
    };


    struct Normal: NormalNode
    {
      static Ref<Node> instantiate(BackRef<ExecutionContext> c)
      {
        return new Normal(c);
      }

      static const NodeType *type;
      const NodeType *getType() const { return type; }

      Normal(BackRef<ExecutionContext> c): Node(c) {}

      const vector<Vector3> &getNormal() const { return normal; }

    private:
      friend void initializeGeopropComponent();

      vector<Vector3> normal; // The X3D specification names this 'vector',
           but that clashes with the std::vector<>

      EventSource normalChanged;

      Time normalStamp;
    };
  }
}

#endif // ARCHON_X3D_GEOPROP_H
```

Listing: **archon/x3d/server/geoprop.C**


```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

/**
 * \file geoprop.C
 *
 * X3D Geometric properties component
 */

#include <archon/x3d/server/field.H>
#include <archon/x3d/server/geoprop.H>

namespace Archon
{
  namespace X3D
  {
    extern long nodeTypeIndexEnumerator;

    const NodeType *GeometricPropertyNode::type = 0;

    const NodeType *ColorNode::type = 0;
    const NodeType *CoordinateNode::type = 0;
    const NodeType *NormalNode::type = 0;

    const NodeType *Color::type = 0;
    const NodeType *ColorRGBA::type = 0;
    const NodeType *Coordinate::type = 0;
    const NodeType *Normal::type = 0;

    void initializeGeopropComponent()
    {
      vector<const FieldBase *> fields;

      // GeometricPropertyNode

      GeometricPropertyNode::type =
        NodeType::newAbstract("GeometricPropertyNode", false, 0,
                              Node::type);


      // ColorNode
```

```
ColorNode::type =
  NodeType::newAbstract("ColorNode", false, 0,
                        GeometricPropertyNode::type);


// CoordinateNode

CoordinateNode::type =
  NodeType::newAbstract("CoordinateNode", false, 0,
                        GeometricPropertyNode::type);


// NormalNode

NormalNode::type =
  NodeType::newAbstract("NormalNode", false, 0,
                        GeometricPropertyNode::type);


// Color

fields.push_back(newExposedField("color", MFColor::type,
                                 &Color::color,
                                 &Color::colorChanged,
                                 &Color::colorStamp));
Color::type =
  NodeType::newConcrete("Color", Color::instantiate, &fields,
                        ColorNode::type);
fields.clear();


// ColorRGBA

fields.push_back(newExposedField("color", MFColorRGBA::type,
                                 &ColorRGBA::color,
                                 &ColorRGBA::colorChanged,
                                 &ColorRGBA::colorStamp));
ColorRGBA::type =
  NodeType::newConcrete("ColorRGBA", ColorRGBA::instantiate, &fields,
                        ColorNode::type);
fields.clear();


// Coordinate

fields.push_back(newExposedField("point", MFVec3f::type,
                                 &Coordinate::point,
                                 &Coordinate::pointChanged,
                                 &Coordinate::pointStamp));
Coordinate::type =
  NodeType::newConcrete("Coordinate", Coordinate::instantiate, &fields,
                        CoordinateNode::type);
fields.clear();


// Normal

fields.push_back(newExposedField("vector", MFVec3f::type,
```



```
                                         &Normal::normal,
                                         &Normal::normalChanged,
                                         &Normal::normalStamp));
      Normal::type =
        NodeType::newConcrete("Normal", Normal::instantiate, &fields,
                              NormalNode::type);
      fields.clear();
    }
  }
}
```

Listing: **archon/x3d/server/group.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_GROUP_H
#define ARCHON_X3D_GROUP_H

/**
 * \file group.H
 *
 * X3D Aggregation and transformation component
 */

#include <vector>
#include <string>

#include <archon/math/vector.H>
#include <archon/math/geometry.H>

#include <archon/x3d/server/core.H>

namespace Archon
{
  namespace X3D
  {
    using namespace Math;


    struct BoundedObject: virtual Node
    {
```

```
  static const NodeType *type;

  const Vector3 &getBboxCenter() const { return bboxCenter; }
  const Vector3 &getBboxSize() const { return bboxSize; }

protected:
  BoundedObject():
    bboxCenter(0, 0, 0), bboxSize(-1, -1, -1) {}

private:
  friend void initializeGroupComponent();

  Vector3 bboxCenter;
  Vector3 bboxSize;
};


struct GroupingNode: virtual ChildNode
{
  static const NodeType *type;
  static const NodeSequenceFieldBase *childrenField;

  typedef vector<Ref<ChildNode> >::iterator       iterator;
  typedef vector<Ref<ChildNode> >::const_iterator const_iterator;

  iterator       begin()       { return children.begin(); }
  const_iterator begin() const { return children.begin(); }
  iterator       end()         { return children.end(); }
  const_iterator end()   const { return children.end(); }

protected:
  GroupingNode() {}

private:
  friend void initializeGroupComponent();

  vector<Ref<ChildNode> > children;

  EventSource childrenChanged;

  Time childrenStamp;

  void addChildren(const vector<Ref<ChildNode> > &);
  void removeChildren(const vector<Ref<ChildNode> > &);
};


struct Group: GroupingNode, BoundedObject
{
  static Ref<Node> instantiate(BackRef<ExecutionContext> c)
  {
    return new Group(c);
  }

  static const NodeType *type;
  const NodeType *getType() const { return type; }

  Group(BackRef<ExecutionContext> c): Node(c) {}
```



```
  friend void initializeGroupComponent();
};


struct Switch: GroupingNode, BoundedObject
{
  static Ref<Node> instantiate(BackRef<ExecutionContext> c)
  {
    return new Switch(c);
  }

  static const NodeType *type;
  const NodeType *getType() const { return type; }

  Switch(BackRef<ExecutionContext> c): Node(c), whichChoice(-1) {}

  int getWhichChoise() const { return whichChoice; }

private:
  friend void initializeGroupComponent();

  int whichChoice;

  EventSource whichChoiceChanged;

  Time whichChoiceStamp;
};


struct Transform: GroupingNode, BoundedObject
{
  static Ref<Node> instantiate(BackRef<ExecutionContext> c)
  {
    return new Transform(c);
  }

  static const NodeType *type;
  const NodeType *getType() const { return type; }

  Transform(BackRef<ExecutionContext> c):
    Node(c),
    center(0, 0, 0),
    rotation(AxisRot3(Vector3(0, 0, 1), 0)),
    scale(1, 1, 1),
    scaleOrientation(AxisRot3(Vector3(0, 0, 1), 0)),
    translation(0, 0, 0) {}

  const Vector3 &getCenter()            const { return center; }
  const AxisRot3 &getRotation()         const { return rotation; }
  const Vector3 &getScale()             const { return scale; }
  const AxisRot3 &getScaleOrientation() const { return scaleOrientation
      ; }
  const Vector3 &getTranslation()       const { return translation; }

private:
  friend void initializeGroupComponent();
```

```
      Vector3 center;
      AxisRot3 rotation;
      Vector3 scale;
      AxisRot3 scaleOrientation;
      Vector3 translation;

      EventSource centerChanged;
      EventSource rotationChanged;
      EventSource scaleChanged;
      EventSource scaleOrientationChanged;
      EventSource translationChanged;

      Time centerStamp;
      Time rotationStamp;
      Time scaleStamp;
      Time scaleOrientationStamp;
      Time translationStamp;
    };
  }
}

#endif // ARCHON_X3D_GROUP_H
```

Listing: **archon/x3d/server/group.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

/**
 * \file group.C
 *
 * X3D Aggregation and transformation component
 */

#include <archon/x3d/server/field_type.H>
#include <archon/x3d/server/field.H>
#include <archon/x3d/server/group.H>
#include <archon/x3d/server/scene.H>

namespace Archon
{
  namespace X3D
```



```
  {
    extern long nodeTypeIndexEnumerator;

    const NodeType *BoundedObject::type = 0;

    const NodeType *GroupingNode::type = 0;
    const NodeSequenceFieldBase *GroupingNode::childrenField = 0;

    const NodeType *Group::type = 0;
    const NodeType *Switch::type = 0;
    const NodeType *Transform::type = 0;


    /**
     * Initialize the runtime type information for the Aggregation and
     * transformation component
     * The following components must be initialized first:
     * - Core Component
     */
    void initializeGroupComponent()
    {
      vector<const FieldBase *> fields;


      // BoundedObject

      fields.push_back(newPrivateField("bboxCenter", SFVec3f::type,
                                       &BoundedObject::bboxCenter));
      fields.push_back(newPrivateField("bboxSize", SFVec3f::type,
                                       &BoundedObject::bboxSize));
      BoundedObject::type =
        NodeType::newAbstract("BoundedObject", false,
                              &fields, Node::type);
      fields.clear();


      // GroupingNode

      fields.push_back(newEventIn("addChildren",
                                  &GroupingNode::addChildren,
                                  &GroupingNode::childrenStamp,
                                  FieldBase::NOVARIANT));
      fields.push_back(newEventIn("removeChildren",
                                  &GroupingNode::removeChildren,
                                  &GroupingNode::childrenStamp,
                                  FieldBase::NOVARIANT));
      GroupingNode::childrenField =
        newExposedField("children",
                        &GroupingNode::children,
                        &GroupingNode::childrenChanged,
                        &GroupingNode::childrenStamp);
      fields.push_back(GroupingNode::childrenField);
      GroupingNode::type =
        NodeType::newAbstract("GroupingNode", false,
                              &fields, ChildNode::type);
      fields.clear();
```

```
    // Group

    Group::type =
      NodeType::newConcrete("Group", Group::instantiate,
                            0, GroupingNode::type, BoundedObject::type);


    // Switch

    fields.push_back(newExposedField("whichChoice", SFInt32::type,
                                     &Switch::whichChoice,
                                     &Switch::whichChoiceChanged,
                                     &Switch::whichChoiceStamp));
    Switch::type =
      NodeType::newConcrete("Switch", Switch::instantiate,
                            &fields, GroupingNode::type, BoundedObject::
                                 type);
    fields.clear();


    // Transform

    fields.push_back(newExposedField("center", SFVec3f::type,
                                     &Transform::center,
                                     &Transform::centerChanged,
                                     &Transform::centerStamp));
    fields.push_back(newExposedField("rotation", SFRotation::type,
                                     &Transform::rotation,
                                     &Transform::rotationChanged,
                                     &Transform::rotationStamp));
    fields.push_back(newExposedField("scale", SFVec3f::type,
                                     &Transform::scale,
                                     &Transform::scaleChanged,
                                     &Transform::scaleStamp));
    fields.push_back(newExposedField("scaleOrientation", SFRotation::type,
                                     &Transform::scaleOrientation,
                                     &Transform::scaleOrientationChanged,
                                     &Transform::scaleOrientationStamp));
    fields.push_back(newExposedField("translation", SFVec3f::type,
                                     &Transform::translation,
                                     &Transform::translationChanged,
                                     &Transform::translationStamp));
    Transform::type =
      NodeType::newConcrete("Transform", Transform::instantiate,
                            &fields, GroupingNode::type, BoundedObject::
                                 type);
    fields.clear();
  }

  void GroupingNode::addChildren(const vector<Ref<ChildNode> > &c)
  {
    for(unsigned i=0; i<c.size(); ++i)
      if(find(children.begin(), children.end(), c[i]) == children.end())
        children.push_back(c[i]);
    Event event(childrenField->get(this), childrenStamp);
    childrenChanged.cascadeEvent(&event);
  }
```



```
    void GroupingNode::removeChildren(const vector<Ref<ChildNode> > &c)
    {
      for(unsigned i=0; i<c.size(); ++i)
        children.erase(std::remove(children.begin(), children.end(),
                                   c[i]), children.end());

      Ref<NodeSequenceValue> val =
        new NodeSequenceValue(MFNode::type);
      vector<Ref<Node> > &v = val->value;
      v.resize(children.size());
      for(unsigned i=0; i<v.size(); ++i) v[i] = children[i];
      Event event(val.get(), childrenStamp);
      childrenChanged.cascadeEvent(&event);
    }
  }
}
```

Listing: **archon/x3d/server/interpolation.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_INTERPOLATION_H
#define ARCHON_X3D_INTERPOLATION_H

/*
 * \file interpolation.H
 *
 *  X3D Interpolation component
 */

#include <archon/math/vector.H>

#include <archon/x3d/server/core.H>
#include <archon/x3d/server/scene.H>

namespace Archon
{
  namespace X3D
  {
    using namespace Math;
```

```
struct InterpolatorNode: virtual ChildNode
{
  static const NodeType *type;

protected:
  InterpolatorNode() {}

  virtual void setFraction(const double &) = 0; // SFFloat [in]
  vector<double> key;                           // MFFloat [in, out]

  Time fractionStamp;

private:
  friend void initializeInterpolationComponent();

  EventSource keyChanged;

  Time keyStamp;
};


struct ColorInterpolator: InterpolatorNode
{
  static Ref<Node> instantiate(BackRef<ExecutionContext> c)
  {
    return new ColorInterpolator(c);
  }

  static const NodeType *type;
  const NodeType *getType() const { return type; }

  ColorInterpolator(BackRef<ExecutionContext> c):
    Node(c), value(Vector3::zero()) {}

private:
  friend void initializeInterpolationComponent();

  vector<Vector3> keyValue; // MFVec3f [in,out]
  Vector3         value;    // SFVec3f [out]

  EventSource keyValueChanged;
  EventSource valueChanged;

  Time keyValueStamp;

  void setFraction(const double &);
};


struct CoordinateInterpolator: InterpolatorNode
{
  static Ref<Node> instantiate(BackRef<ExecutionContext> c)
  {
    return new CoordinateInterpolator(c);
  }

  static const NodeType *type;
```



```
  const NodeType *getType() const { return type; }

  CoordinateInterpolator(BackRef<ExecutionContext> c):
    Node(c) {}

private:
  friend void initializeInterpolationComponent();

  vector<Vector3> keyValue; // MFVec3f [in,out]
  vector<Vector3> value;    // MFVec3f [out]

  EventSource keyValueChanged;
  EventSource valueChanged;

  Time keyValueStamp;

  void setFraction(const double &);
};


struct CoordinateInterpolator2D: InterpolatorNode
{
  static Ref<Node> instantiate(BackRef<ExecutionContext> c)
  {
    return new CoordinateInterpolator2D(c);
  }

  static const NodeType *type;
  const NodeType *getType() const { return type; }

  CoordinateInterpolator2D(BackRef<ExecutionContext> c):
    Node(c) {}

private:
  friend void initializeInterpolationComponent();

  vector<Vector2> keyValue; // MFVec2f [in,out]
  vector<Vector2> value;    // MFVec2f [out]

  EventSource keyValueChanged;
  EventSource valueChanged;

  Time keyValueStamp;

  void setFraction(const double &);
};


struct NormalInterpolator: InterpolatorNode
{
  static Ref<Node> instantiate(BackRef<ExecutionContext> c)
  {
    return new NormalInterpolator(c);
  }

  static const NodeType *type;
  const NodeType *getType() const { return type; }
```

```
  NormalInterpolator(BackRef<ExecutionContext> c): Node(c) {}

private:
  friend void initializeInterpolationComponent();

  vector<Vector3> keyValue; // MFVec3f [in,out]
  vector<Vector3> value;    // MFVec3f [out]

  EventSource keyValueChanged;
  EventSource valueChanged;

  Time keyValueStamp;

  void setFraction(const double &);
};


struct OrientationInterpolator: InterpolatorNode
{
  static Ref<Node> instantiate(BackRef<ExecutionContext> c)
  {
    return new OrientationInterpolator(c);
  }

  static const NodeType *type;
  const NodeType *getType() const { return type; }

  OrientationInterpolator(BackRef<ExecutionContext> c):
    Node(c), value(Vector3(0, 0, 1), 0) {}

private:
  friend void initializeInterpolationComponent();

  vector<AxisRot3> keyValue; // MFRotation [in,out]
  AxisRot3         value;    // SFRotation [out]

  EventSource keyValueChanged;
  EventSource valueChanged;

  Time keyValueStamp;

  void setFraction(const double &);
};


struct PositionInterpolator: InterpolatorNode
{
  static Ref<Node> instantiate(BackRef<ExecutionContext> c)
  {
    return new PositionInterpolator(c);
  }

  static const NodeType *type;
  const NodeType *getType() const { return type; }

  PositionInterpolator(BackRef<ExecutionContext> c):
    Node(c), value(Vector3::zero()) {}
```



```
private:
  friend void initializeInterpolationComponent();

  vector<Vector3> keyValue; // MFVec3f [in,out]
  Vector3         value;    // SFVec3f [out]

  EventSource keyValueChanged;
  EventSource valueChanged;

  Time keyValueStamp;

  void setFraction(const double &);
};


struct PositionInterpolator2D: InterpolatorNode
{
  static Ref<Node> instantiate(BackRef<ExecutionContext> c)
  {
    return new PositionInterpolator2D(c);
  }

  static const NodeType *type;
  const NodeType *getType() const { return type; }

  PositionInterpolator2D(BackRef<ExecutionContext> c):
    Node(c), value(Vector3::zero()) {}

private:
  friend void initializeInterpolationComponent();

  vector<Vector2> keyValue; // MFVec2f [in,out]
  Vector2         value;    // SFVec2f [out]

  EventSource keyValueChanged;
  EventSource valueChanged;

  Time keyValueStamp;

  void setFraction(const double &);
};


struct ScalarInterpolator: InterpolatorNode
{
  static Ref<Node> instantiate(BackRef<ExecutionContext> c)
  {
    return new ScalarInterpolator(c);
  }

  static const NodeType *type;
  const NodeType *getType() const { return type; }

  ScalarInterpolator(BackRef<ExecutionContext> c):
    Node(c), value(0) {}

private:
  friend void initializeInterpolationComponent();
```

```
      vector<double> keyValue; // MFDouble [in,out]
      double         value;    // SFDouble [out]

      EventSource keyValueChanged;
      EventSource valueChanged;

      Time keyValueStamp;

      void setFraction(const double &);
    };
  }
}

#endif // ARCHON_X3D_INTERPOLATION_H
```

Listing: archon/x3d/server/interpolation.C

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

/*
 * \file time.C
 *
 *  X3D Time component
 */

#include <archon/math/functions.H>
#include <archon/util/color.H>

#include <archon/x3d/server/field_type.H>
#include <archon/x3d/server/field.H>
#include <archon/x3d/server/interpolation.H>

namespace Archon
{
  namespace X3D
  {
    extern long nodeTypeIndexEnumerator;

    const NodeType *InterpolatorNode::type = 0;
    const NodeType *ColorInterpolator::type = 0;
```



```
    const NodeType *CoordinateInterpolator::type = 0;
    const NodeType *CoordinateInterpolator2D::type = 0;
    const NodeType *NormalInterpolator::type = 0;
    const NodeType *OrientationInterpolator::type = 0;
    const NodeType *PositionInterpolator::type = 0;
    const NodeType *PositionInterpolator2D::type = 0;
    const NodeType *ScalarInterpolator::type = 0;

    void initializeInterpolationComponent()
    {
      vector<const FieldBase *> fields;


      // InterpolatorNode

      fields.push_back(newEventIn("fraction", SFFloat::type,
                                  &InterpolatorNode::setFraction,
                                  &InterpolatorNode::fractionStamp));
      fields.push_back(newExposedField("key", MFFloat::type,
                                       &InterpolatorNode::key,
                                       &InterpolatorNode::keyChanged,
                                       &InterpolatorNode::keyStamp));
      InterpolatorNode::type =
        NodeType::newAbstract("InterpolatorNode", false, &fields,
                              ChildNode::type);
      fields.clear();


      // ColorInterpolator

      fields.push_back(newExposedField("keyValue", MFColor::type,
                                       &ColorInterpolator::keyValue,
                                       &ColorInterpolator::keyValueChanged,
                                       &ColorInterpolator::keyValueStamp));
      fields.push_back(newEventOut("value", SFColor::type,
                                   &ColorInterpolator::value,
                                   &ColorInterpolator::valueChanged));
      ColorInterpolator::type =
        NodeType::newConcrete("ColorInterpolator", ColorInterpolator::
             instantiate, &fields,
                              InterpolatorNode::type);
      fields.clear();


      // CoordinateInterpolator

      fields.push_back(newExposedField("keyValue", MFVec3f::type,
                                       &CoordinateInterpolator::keyValue,
                                       &CoordinateInterpolator::
                                            keyValueChanged,
                                       &CoordinateInterpolator::keyValueStamp
                                            ));
      fields.push_back(newEventOut("value", MFVec3f::type,
                                   &CoordinateInterpolator::value,
                                   &CoordinateInterpolator::valueChanged));
      CoordinateInterpolator::type =
        NodeType::newConcrete("CoordinateInterpolator",
             CoordinateInterpolator::instantiate, &fields,
```

```
                        InterpolatorNode::type);
fields.clear();


// CoordinateInterpolator2D

fields.push_back(newExposedField("keyValue", MFVec2f::type,
                                 &CoordinateInterpolator2D::keyValue,
                                 &CoordinateInterpolator2D::
                                      keyValueChanged,
                                 &CoordinateInterpolator2D::
                                      keyValueStamp));
fields.push_back(newEventOut("value", MFVec2f::type,
                             &CoordinateInterpolator2D::value,
                             &CoordinateInterpolator2D::valueChanged));
CoordinateInterpolator2D::type =
  NodeType::newConcrete("CoordinateInterpolator2D",
       CoordinateInterpolator2D::instantiate, &fields,
                        InterpolatorNode::type);
fields.clear();


// NormalInterpolator

fields.push_back(newExposedField("keyValue", MFVec3f::type,
                                 &NormalInterpolator::keyValue,
                                 &NormalInterpolator::keyValueChanged,
                                 &NormalInterpolator::keyValueStamp));
fields.push_back(newEventOut("value", MFVec3f::type,
                             &NormalInterpolator::value,
                             &NormalInterpolator::valueChanged));
NormalInterpolator::type =
  NodeType::newConcrete("NormalInterpolator", NormalInterpolator::
       instantiate, &fields,
                        InterpolatorNode::type);
fields.clear();


// OrientationInterpolator

fields.push_back(newExposedField("keyValue", MFRotation::type,
                                 &OrientationInterpolator::keyValue,
                                 &OrientationInterpolator::
                                      keyValueChanged,
                                 &OrientationInterpolator::
                                      keyValueStamp));
fields.push_back(newEventOut("value", SFRotation::type,
                             &OrientationInterpolator::value,
                             &OrientationInterpolator::valueChanged));
OrientationInterpolator::type =
  NodeType::newConcrete("OrientationInterpolator",
       OrientationInterpolator::instantiate, &fields,
                        InterpolatorNode::type);
fields.clear();


// PositionInterpolator
```



```
  fields.push_back(newExposedField("keyValue", MFVec3f::type,
                                   &PositionInterpolator::keyValue,
                                   &PositionInterpolator::keyValueChanged
                                        ,
                                   &PositionInterpolator::keyValueStamp
                                        ));
  fields.push_back(newEventOut("value", SFVec3f::type,
                               &PositionInterpolator::value,
                               &PositionInterpolator::valueChanged));
  PositionInterpolator::type =
    NodeType::newConcrete("PositionInterpolator", PositionInterpolator::
         instantiate, &fields,
                          InterpolatorNode::type);
  fields.clear();


  // PositionInterpolator2D

  fields.push_back(newExposedField("keyValue", MFVec2f::type,
                                   &PositionInterpolator2D::keyValue,
                                   &PositionInterpolator2D::
                                        keyValueChanged,
                                   &PositionInterpolator2D::keyValueStamp
                                        ));
  fields.push_back(newEventOut("value", SFVec2f::type,
                               &PositionInterpolator2D::value,
                               &PositionInterpolator2D::valueChanged));
  PositionInterpolator2D::type =
    NodeType::newConcrete("PositionInterpolator2D",
         PositionInterpolator2D::instantiate, &fields,
                          InterpolatorNode::type);
  fields.clear();


  // ScalarInterpolator

  fields.push_back(newExposedField("keyValue", MFFloat::type,
                                   &ScalarInterpolator::keyValue,
                                   &ScalarInterpolator::keyValueChanged,
                                   &ScalarInterpolator::keyValueStamp));
  fields.push_back(newEventOut("value", SFFloat::type,
                               &ScalarInterpolator::value,
                               &ScalarInterpolator::valueChanged));
  ScalarInterpolator::type =
    NodeType::newConcrete("ScalarInterpolator", ScalarInterpolator::
         instantiate, &fields,
                          InterpolatorNode::type);
  fields.clear();
}

/**
 * \todo Huh, this is heavy, could it be done in an other more
 * afficient way? Maybe using quaternions.
 *
 * Nothe that the outer-most linear interpolation serves only to
 * soften the transistion from y1 to y2. Without it numerical
 * error has too big an impact on the visual result.
 */
```

```
Vector3 normalInterp(double x, double x1, double x2, Vector3 y1, Vector3
     y2)
{
  y1.normalize();
  y2.normalize();
  double p = dot(y1, y2);
  if(p==1) return y1;
  Vector3 n = p==-1 ? Vector3(0, 0, 1) : y1 * y2;
  return
    linInterp(x, x1, x2,
              Matrix3x3(AxisRot3(n, linInterp(x, x1, x2, 0.0, acos(p))))(
                   y1),
              Matrix3x3(AxisRot3(n, linInterp(x, x1, x2, -acos(p
                   ), 0.0)))(y2));
}

/**
 * \todo Is this the right way to interpolate between
 * orientations. It may not be what the X3D spec. expects. An
 * alternative sollution would be based on interpolation on
 * quaternions.
 */
AxisRot3 rotInterp(double x, double x1, double x2, AxisRot3 y1, AxisRot3
     y2)
{
  // Pick the shortest arc
  double d = y2.angle - y1.angle;
  if(d < -M_PI) y2.angle += 2*M_PI;
  else if(d > M_PI) y1.angle += 2*M_PI;

  return AxisRot3(normalInterp(x, x1, x2, y1.normalizedAxis, y2.
       normalizedAxis), linInterp(x, x1, x2, y1.angle, y2.angle));
}

void ColorInterpolator::setFraction(const double &f)
{
  unsigned i=0;
  unsigned n=min(key.size(), keyValue.size());
  while(i<n && key[i]<=f) ++i;
  value = i == n ?
    n ? keyValue[n-1] : Vector3::zero() :
    i ? Color::interp(f, key[i-1], key[i], keyValue[i-1], keyValue[i]) :
    keyValue[0];
  Event event(new SimpleValue<Vector3>(SFColor::type, value),
       fractionStamp);
  valueChanged.cascadeEvent(&event);
}

void CoordinateInterpolator::setFraction(const double &f)
{
  unsigned i=0;
  unsigned n=key.size();
  unsigned m=keyValue.size()/key.size();
  while(i<n && key[i]<=f) ++i;
  if(i==n)
  {
    if(n)
    {
```



```
      vector<Vector3>::iterator j = keyValue.begin() + n*m;
      value.assign(j-m, j);
    }
    else value.clear();
  }
  else
  {
    if(i)
    {
      value.resize(m);
      vector<Vector3>::iterator j = keyValue.begin() + i*m;
      for(unsigned k=0; k<m; ++j, ++k)
        value[k] = linInterp(f, key[i-1], key[i], *(j-m), *j);
    }
    else
    {
      vector<Vector3>::iterator j = keyValue.begin();
      value.assign(j, j+m);
    }
  }
  Event event(new SimpleValue<vector<Vector3> >(MFVec3f::type, value),
       fractionStamp);
  valueChanged.cascadeEvent(&event);
}

void CoordinateInterpolator2D::setFraction(const double &f)
{
  unsigned i=0;
  unsigned n=key.size();
  unsigned m=keyValue.size()/key.size();
  while(i<n && key[i]<=f) ++i;
  if(i==n)
  {
    if(n)
    {
      vector<Vector2>::iterator j = keyValue.begin() + n*m;
      value.assign(j-m, j);
    }
    else value.clear();
  }
  else
  {
    if(i)
    {
      value.resize(m);
      vector<Vector2>::iterator j = keyValue.begin() + i*m;
      for(unsigned k=0; k<m; ++j, ++k)
        value[k] = linInterp(f, key[i-1], key[i], *(j-m), *j);
    }
    else
    {
      vector<Vector2>::iterator j = keyValue.begin();
      value.assign(j, j+m);
    }
  }
  Event event(new SimpleValue<vector<Vector2> >(MFVec2f::type, value),
       fractionStamp);
  valueChanged.cascadeEvent(&event);
```

```
}

void NormalInterpolator::setFraction(const double &f)
{
  unsigned i=0;
  unsigned n=key.size();
  unsigned m=keyValue.size()/key.size();
  while(i<n && key[i]<=f) ++i;
  if(i==n)
  {
    if(n)
    {
      vector<Vector3>::iterator j = keyValue.begin() + n*m;
      value.assign(j-m, j);
    }
    else value.clear();
  }
  else
  {
    if(i)
    {
      value.resize(m);
      vector<Vector3>::iterator j = keyValue.begin() + i*m;
      for(unsigned k=0; k<m; ++j, ++k)
        value[k] = normalInterp(f, key[i-1], key[i], *(j-m), *j);
    }
    else
    {
      vector<Vector3>::iterator j = keyValue.begin();
      value.assign(j, j+m);
    }
  }
  Event event(new SimpleValue<vector<Vector3> >(MFVec3f::type, value),
       fractionStamp);
  valueChanged.cascadeEvent(&event);
}

void OrientationInterpolator::setFraction(const double &f)
{
  unsigned i=0;
  unsigned n=min(key.size(), keyValue.size());
  while(i<n && key[i]<=f) ++i;
  value = i == n ?
    n ? keyValue[n-1] : AxisRot3(Vector3(0, 0, 1), 0) :
    i ? rotInterp(f, key[i-1], key[i], keyValue[i-1], keyValue[i]) :
    keyValue[0];
  Event event(new SimpleValue<AxisRot3>(SFRotation::type, value),
       fractionStamp);
  valueChanged.cascadeEvent(&event);
}

void PositionInterpolator::setFraction(const double &f)
{
  unsigned i=0;
  unsigned n=min(key.size(), keyValue.size());
  while(i<n && key[i]<=f) ++i;
  value = i == n ?
    n ? keyValue[n-1] : Vector3::zero() :
```



```
      i ? linInterp(f, key[i-1], key[i], keyValue[i-1], keyValue[i]) :
      keyValue[0];
    Event event(new SimpleValue<Vector3>(SFVec3f::type, value),
         fractionStamp);
    valueChanged.cascadeEvent(&event);
  }

  void PositionInterpolator2D::setFraction(const double &f)
  {
    unsigned i=0;
    unsigned n=min(key.size(), keyValue.size());
    while(i<n && key[i]<=f) ++i;
    value = i == n ?
      n ? keyValue[n-1] : Vector2::zero() :
      i ? linInterp(f, key[i-1], key[i], keyValue[i-1], keyValue[i]) :
      keyValue[0];
    Event event(new SimpleValue<Vector2>(SFVec2f::type, value),
         fractionStamp);
    valueChanged.cascadeEvent(&event);
  }

  void ScalarInterpolator::setFraction(const double &f)
  {
    unsigned i=0;
    unsigned n=min(key.size(), keyValue.size());
    while(i<n && key[i]<=f) ++i;
    value = i == n ?
      n ? keyValue[n-1] : 0 :
      i ? linInterp(f, key[i-1], key[i], keyValue[i-1], keyValue[i]) :
      keyValue[0];
    Event event(new SimpleValue<double>(SFFloat::type, value),
         fractionStamp);
    valueChanged.cascadeEvent(&event);
  }
 }
}
```

Listing: **archon/x3d/server/light.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */
```

```
#ifndef ARCHON_X3D_LIGHT_H
#define ARCHON_X3D_LIGHT_H

/*
 * \file light.H
 *
 *  X3D Lighting component
 */

#include <archon/math/vector.H>

#include <archon/x3d/server/core.H>
#include <archon/x3d/server/scene.H>

namespace Archon
{
  namespace X3D
  {
    using namespace Math;


    struct LightNode: virtual ChildNode
    {
      static const NodeType *type;

      double getAmbientIntensity() const { return ambientIntensity; }
      const Vector3 &getColor()    const { return color; }
      double getIntensity()        const { return intensity; }
      bool getOn()                 const { return on; }

    protected:
      LightNode():
        ambientIntensity(0), color(1, 1, 1),
        intensity(1), on(true) {}

    private:
      friend void initializeLightComponent();

      double  ambientIntensity;
      Vector3 color;
      double  intensity;
      bool    on;

      EventSource ambientIntensityChanged;
      EventSource colorChanged;
      EventSource intensityChanged;
      EventSource onChanged;

      Time ambientIntensityStamp;
      Time colorStamp;
      Time intensityStamp;
      Time onStamp;
    };


    struct DirectionalLight: LightNode
    {
      static Ref<Node> instantiate(BackRef<ExecutionContext> c)
```



```
      {
        return new DirectionalLight(c);
      }

      static const NodeType *type;
      const NodeType *getType() const { return type; }

      DirectionalLight(BackRef<ExecutionContext> c):
        Node(c),
        direction(0, 0, -1) {}

      const Vector3 &getDirection() const { return direction; }

    private:
      friend void initializeLightComponent();

      Vector3 direction;

      EventSource directionChanged;

      Time directionStamp;
    };


    struct PointLight: LightNode
    {
      static Ref<Node> instantiate(BackRef<ExecutionContext> c)
      {
        return new PointLight(c);
      }

      static const NodeType *type;
      const NodeType *getType() const { return type; }

      PointLight(BackRef<ExecutionContext> c):
        Node(c),
        attenuation(1, 0, 0),
        location(0, 0, 0),
        radius(100) {}

      const Vector3 &getAttenuation() const { return attenuation; }
      const Vector3 &getLocation()    const { return location; }
      double getRadius()              const { return radius; }

    private:
      friend void initializeLightComponent();

      Vector3 attenuation;
      Vector3 location;
      double  radius;

      EventSource attenuationChanged;
      EventSource locationChanged;
      EventSource radiusChanged;

      Time attenuationStamp;
      Time locationStamp;
      Time radiusStamp;
```

```
  };


  struct SpotLight: LightNode
  {
    static Ref<Node> instantiate(BackRef<ExecutionContext> c)
    {
      return new SpotLight(c);
    }

    static const NodeType *type;
    const NodeType *getType() const { return type; }

    SpotLight(BackRef<ExecutionContext> c):
      Node(c),
      attenuation(1, 0, 0),
      beamWidth(1.570796),
      cutOffAngle(0.785398),
      direction(0, 0, -1),
      location(0, 0, 0),
      radius(100) {}

    const  Vector3 &getAttenuation() const { return attenuation; }
    double getBeamWidth()            const { return beamWidth; }
    double getCutOffAngle()          const { return cutOffAngle; }
    const  Vector3 &getDirection()   const { return direction; }
    const  Vector3 &getLocation()    const { return location; }
    double getRadius()               const { return radius; }

  private:
    friend void initializeLightComponent();

    Vector3 attenuation;
    double  beamWidth;
    double  cutOffAngle;
    Vector3 direction;
    Vector3 location;
    double  radius;

    EventSource attenuationChanged;
    EventSource beamWidthChanged;
    EventSource cutOffAngleChanged;
    EventSource directionChanged;
    EventSource locationChanged;
    EventSource radiusChanged;

    Time attenuationStamp;
    Time beamWidthStamp;
    Time cutOffAngleStamp;
    Time directionStamp;
    Time locationStamp;
    Time radiusStamp;
  };
 }
}

#endif // ARCHON_X3D_LIGHT_H
```


Listing: **archon/x3d/server/light.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

/*
 * \file light.C
 *
 *  X3D Lighting component
 */

#include <archon/x3d/server/field_type.H>
#include <archon/x3d/server/field.H>
#include <archon/x3d/server/light.H>

namespace Archon
{
  namespace X3D
  {
    extern long nodeTypeIndexEnumerator;

    const NodeType *LightNode::type = 0;
    const NodeType *DirectionalLight::type = 0;
    const NodeType *PointLight::type = 0;
    const NodeType *SpotLight::type = 0;

    void initializeLightComponent()
    {
      vector<const FieldBase *> fields;


      // LightNode

      fields.push_back(newExposedField("ambientIntensity", SFFloat::type,
                                       &LightNode::ambientIntensity,
                                       &LightNode::ambientIntensityChanged,
                                       &LightNode::ambientIntensityStamp));
      fields.push_back(newExposedField("color", SFColor::type,
                                       &LightNode::color,
                                       &LightNode::colorChanged,
                                       &LightNode::colorStamp));
      fields.push_back(newExposedField("intensity", SFFloat::type,
                                       &LightNode::intensity,
```

```
                                  &LightNode::intensityChanged,
                                  &LightNode::intensityStamp));
      fields.push_back(newExposedField("on", SFBool::type,
                                  &LightNode::on,
                                  &LightNode::onChanged,
                                  &LightNode::onStamp));
      LightNode::type =
        NodeType::newAbstract("LightNode", false, &fields,
                              ChildNode::type);
      fields.clear();


      // DirectionalLight

      fields.push_back(newExposedField("direction", SFVec3f::type,
                                  &DirectionalLight::direction,
                                  &DirectionalLight::directionChanged,
                                  &DirectionalLight::directionStamp));
      DirectionalLight::type =
        NodeType::newConcrete("DirectionalLight", DirectionalLight::
             instantiate, &fields,
                              LightNode::type);
      fields.clear();


      // PointLight

      fields.push_back(newExposedField("attenuation", SFVec3f::type,
                                  &PointLight::attenuation,
                                  &PointLight::attenuationChanged,
                                  &PointLight::attenuationStamp));
      fields.push_back(newExposedField("location", SFVec3f::type,
                                  &PointLight::location,
                                  &PointLight::locationChanged,
                                  &PointLight::locationStamp));
      fields.push_back(newExposedField("radius", SFFloat::type,
                                  &PointLight::radius,
                                  &PointLight::radiusChanged,
                                  &PointLight::radiusStamp));
      PointLight::type =
        NodeType::newConcrete("PointLight", PointLight::instantiate, &fields,
                              LightNode::type);
      fields.clear();


      // SpotLight

      fields.push_back(newExposedField("attenuation", SFVec3f::type,
                                  &SpotLight::attenuation,
                                  &SpotLight::attenuationChanged,
                                  &SpotLight::attenuationStamp));
      fields.push_back(newExposedField("beamWidth", SFFloat::type,
                                  &SpotLight::beamWidth,
                                  &SpotLight::beamWidthChanged,
                                  &SpotLight::beamWidthStamp));
      fields.push_back(newExposedField("cutOffAngle", SFFloat::type,
                                  &SpotLight::cutOffAngle,
                                  &SpotLight::cutOffAngleChanged,
```



```
                                  &SpotLight::cutOffAngleStamp));
      fields.push_back(newExposedField("direction", SFVec3f::type,
                                  &SpotLight::direction,
                                  &SpotLight::directionChanged,
                                  &SpotLight::directionStamp));
      fields.push_back(newExposedField("location", SFVec3f::type,
                                  &SpotLight::location,
                                  &SpotLight::locationChanged,
                                  &SpotLight::locationStamp));
      fields.push_back(newExposedField("radius", SFFloat::type,
                                  &SpotLight::radius,
                                  &SpotLight::radiusChanged,
                                  &SpotLight::radiusStamp));
      SpotLight::type =
        NodeType::newConcrete("SpotLight", SpotLight::instantiate, &fields,
                              LightNode::type);
      fields.clear();
    }
  }
}
```

Listing: **archon/x3d/server/load.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_LOAD_H
#define ARCHON_X3D_LOAD_H

#include <string>

#include <archon/util/logger.H>
#include <archon/util/web_client.H>
#include <archon/util/job.H>
#include <archon/util/image.H>

#include <archon/x3d/server/core.H>
#include <archon/x3d/server/group.H>

namespace Archon
{
  namespace X3D
```

```
{
  using namespace std;
  using namespace Utilities;

  struct Server;

  /**
   * Manage URI content loading in an asynchronious manner.
   *
   * This class is responsible for loading URI contents for those
   * X3D nodes that have URI fields. The loading is done
   * asynchroniously with respect to the code that utilizes the
   * Loader. That is, each submitted load job is executed in an
   * independant loader thread. For some URI types such as 'image'
   * the loaded contents is streamed directly into the appropriate
   * parser which runs in the context of the associated loader
   * thread. In X3D a URI field often contains multiple alternative
   * URIs. All the URIs are passed to a single loader thread which
   * then attemps to load them one at a time untill success. Loaded
   * contents is represented by a polymorphic contents object type
   * 'Contents' regardless of the real type of contents.
   *
   * Caching: The loader also maintains a memory cache. For each URI
   * that is about to be loaded, the loader first consults the
   * cache. If ther is a 'hit' (based on absolute URI), then the
   * cached contents object is used, and the job of loading is
   * complete. If there it no hit, then the URI is requested, and
   * when it is loaded and possibly parsed, the resulting contents
   * object is added to the cache.
   *
   * Notification: The loader job supports notification upon
   * completion through the virtual method 'onCompletion' (this
   * includes the unsuccessfull jobs too).
   */
  struct Loader: virtual RefObjectBase
  {
    struct Exception: virtual X3D::Exception {};
    struct ArgumentException: Exception, X3D::ArgumentException
    {
      ArgumentException(string l, string m): Utilities::Exception(l, m) {}
    };
    struct ResourceException: Exception, X3D::ResourceException
    {
      ResourceException() {}
      ResourceException(string l, string m): Utilities::Exception(l, m) {}
    };
    struct StateException: Exception, X3D::StateException
    {
      StateException(string l, string m): Utilities::Exception(l, m) {}
    };
    struct InternalException: Exception, X3D::InternalException
    {
      InternalException(string l, string m): Utilities::Exception(l, m) {}
    };

    /**
     * A 'UriType' value is used to indicate what type of contents
     * object that we want. This is also a choise of which kind of
```



```
     * parser to stream the loaded data into.
     */
    enum UriType
    {
      uriType_X3D,
      uriType_Image,
      uriType_Movie,
      uriType_Script
    };

    /**
     * Base class for all types of contents objects. That is,
     * objects holding downloaded/cached URI contents.
     */
    struct Contents: virtual RefObjectBase
    {
      /**
       * Provides unique identifiers over long periods of time.
       *
       * That is, if you ask for a 'personal' identifier (using the
       * 'next' method), then it is guaranteed that your identifier
       * will be different from all other identifiers that have been
       * provided in the past, as well as from all those that will
       * be provided in the future. All right, this is utopia, but
       * we should think of it that way, and if we encounter
       * problems keeping this guarantee then we should raise the
       * number of bits in the enumerator, or think of other more
       * dynamic ways of expanding the enumeration range. CORBA
       * delas with this problem too in its IORs, and requires not
       * only that they be unique during the life of a process, but
       * that the be unique at all times and across the whole wide
       * web.
       *
       * The motivation for this kind of identifier comes from the
       * need to discover equality or change of contents between
       * several parties. The trick is that it is usually much
       * faster to compare two identifiers that to compare the
       * objects/entities that they identify. It is only possible
       * though, if we can guarantee uniqueness over long periods of
       * time as described above.
       *
       * Note that the validity of the asumpsion of uniqueness has
       * implications on security also. If two identifiers clash but
       * have different privileges associated with them, then we
       * have no security.
       */
      struct Id
      {
        Id(): id(0) {}

        static Id next()
        {
          static unsigned long counter = 0;
          return Id(++counter);
        }

        operator bool() { return id; }
        bool operator<(const Id &i) const { return id<i.id; }
```

```
  private:
    Id(unsigned long id): id(id) {}
    unsigned long id;
  };

  Contents(): id(Id::next()) {}

  Id getId() const { return id; }

private:
  Id id;
};

struct X3DContents: Contents
{
  Ref<ExecutionContext> context;
  X3DContents(Ref<ExecutionContext> context): context(context) {}
};

struct ImageContents: Contents
{
  Image image;
  ImageContents(const Image &image): image(image) {}
};

/**
 * \todo Do it!
 */
struct MovieContents: Contents
{
};

struct ScriptContents: Contents
{
  string language;
  string sourceCode;
  string sourceName; // File name
  int sourceLine;    // line number in file name or -1 if unknown
  ScriptContents(string language, string sourceCode,
                 string sourceName, int sourceLine):
    language(language), sourceCode(sourceCode),
    sourceName(sourceName), sourceLine(sourceLine) {}
};

struct FileServer: virtual BackRefObjectBase
{
  struct RequestException: Loader::ResourceException
  {
    RequestException(string l, string m): Utilities::Exception(l, m) {}
  };

  /**
   * Must be re-entrant.
   */
  virtual Ref<Stream::Reader> request(string path)
    throw(RequestException) = 0;
```



```
  string getFileServerId()
  {
    return id;
  }

  FileServer(): id(getNextId()) {}

private:
  string id;

  static string getNextId();
};

/**
 * This class represents one request for a set of alternative
 * URIs. You must instantiate it yourself and submit the objects
 * through Loader::submitJob. Note that it is a reference
 * counted object, and thus should primarily be refered to with
 * Ref<Loader> references. You may use the class "as is" or
 * derive it if you want to receive completion notification.
 *
 * Note that protected derivation is required because
 * RefObjectBase is a virtual base class of Utilities::Job,
 * meaning that classes derived from Loader::Job need to be able
 * to access the constructor of Utilities::Job.
 *
 * \sa Loader::onCompletion
 */
struct Job: Utilities::Job
{
  /**
   * \param fileServer If not null this refers to a server that
   * will handle file-scheme URIs.
   */
  Job(Ref<Loader>, Uri baseUri,
      const vector<string> &uriList, UriType uriType,
      Ref<FileServer> fileServer=0);

  virtual ~Job();

  /**
   * Return true if this job is completed either successfully or
   * with failiure.
   */
  bool isCompleted() { return completed; }

  /**
   * Wait for this job to complete either successfully or with
   * failiure.
   */
  void wait();

  /**
   * Send a termination request to this job. It does not wait
   * for the thread to terminate though.
   */
  void kill();
```

```
  /**
   * Returns null if the job is not completet yet or if it
   * failed to load anything. To determine if the job is still
   * running you must either call 'isCompleted' or 'wait' or
   * override the onCompletion method.
   */
  Ref<const Contents> getContents();

protected:
  /**
   * Called by the loading thread to indicate completion.
   * Override this one if you want to receive the notification.
   */
  virtual void onCompletion() {}

private:
  friend struct Loader;

  Ref<const Contents> loadX3D(Ref<Stream::Reader>, string contentType,
                              Uri sourceUri, int sourceLine, Logger *);
  Ref<const Contents> loadImage(Ref<Stream::Reader>, string contentType
       ,
                                Uri sourceUri, Logger *);
  Ref<const Contents> loadMovie(Ref<Stream::Reader>, string contentType
       ,
                                Uri sourceUri, Logger *);
  Ref<const Contents> loadScript(Ref<Stream::Reader>, string
       contentType,
                                 Uri sourceUri, int sourceLine, Logger
                                       *);
  Ref<const Contents> load(Ref<Stream::Reader>, string contentType,
                           Uri sourceUri, int sourceLine, Logger *);
  void main();

  Ref<Loader> loader;
  Ref<Web::Client> webClient;
  Uri baseUri;
  vector<string> uriList;
  UriType uriType;
  Ref<const Contents> contents;
  bool completed;
  Ref<FileServer> fileServer;
};

~Loader();

Loader(Ref<Server>, Ref<Web::Client>);

/**
 * Submit a new download job. You must instantiate the jobs your
 * self.
 *
 * Do not submit a specific job object more than once.
 *
 * \sa Loader::Job
 */
void submitJob(Ref<Job>);
```



```
      const BackRef<Server> server;

      Ref<Web::Client> getWebClient() const { return webClient; }

    private:
      Logger *logger;
      Ref<Web::Client> webClient;
      Utilities::Job::Queue jobQueue;

      /**
       * \todo We need a thread whos job is tho periodically check the
       * cache for expired entries and remove them
       */
      map<string, Ref<const Contents> > cache; // URIs are the keys

      Mutex mutex;
      Condition cacheChange;
      Condition jobCompletion;
    };
  }
}

#endif // ARCHON_X3D_LOAD_H
```

Listing: **archon/x3d/server/load.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#include <math.h>

#include <string>
#include <vector>

#include <archon/util/text.H>
#include <archon/util/stream.H>
#include <archon/util/pipe.H>
#include <archon/util/image.H>

//#include <archon/x3d/server/parse_vrml.H>
#include <archon/x3d/server/parse_xml.H>
```

```
#include <archon/x3d/server/load.H>
#include <archon/x3d/server/server.H>

using namespace std;

namespace Archon
{
  using namespace Math;
  using namespace Utilities;

  namespace X3D
  {
    namespace
    {
      long sourceNameCounter(string sourceName)
      {
        static map<string, long> m;
        return m[sourceName]++;
      };

      struct PrefixLogger: Utilities::Logger
      {
        string prefix;
        Logger *logger;
        PrefixLogger(string prefix, Logger *logger):
          prefix(prefix), logger(logger) {}

        void log(string message) throw()
        {
          logger->log(prefix + message);
        }

        virtual ~PrefixLogger()
        {{ // The extra scope is needed to work around gcc3.2 bug #8287
        }}
      };
    }

    struct ImageLoadTracker: Image::LoadTracker
    {
      void set(Image *i)
      {
        //progressTracker->set(i);
      }

      void progress(long chars_loaded)
      {
        //progressTracker->progress(chars_loaded);
      }
    };

    string Loader::FileServer::getNextId()
    {
      static Mutex m;
      static unsigned long n = 0;
      unsigned long v;
      {
        Mutex::Lock l(m);
```



```
        v = n++;
      }
      string s;
      for(unsigned i=0; i<8; ++i)
      {
        int d = int(v % 36ul);
        s = (d < 10 ? char('0'+d) : char('A'+(d-10))) + s;
        v /= 36ul;
      }
      return s;
    }

    Loader::Job::Job(Ref<Loader> loader, Uri baseUri,
                     const vector<string> &uriList, UriType uriType,
                     Ref<FileServer> fileServer):
      loader(loader), webClient(loader->webClient), baseUri(baseUri),
      uriList(uriList), uriType(uriType), completed(false),
      fileServer(fileServer)
    {
    }

    Loader::Job::~Job()
    {{ // The extra scope is needed to work around gcc3.2 bug #8287
    }}

    void Loader::Job::wait()
    {
      Mutex::Lock l(loader->mutex);
      while(!completed) loader->jobCompletion.wait();
    }

    void Loader::Job::kill()
    {
      Mutex::Lock l(loader->mutex);
      loader->jobQueue.cancel(this);
    }

    Ref<const Loader::Contents> Loader::Job::getContents()
    {
      Mutex::Lock l(loader->mutex);
      return contents;
    }

    Ref<const Loader::Contents> Loader::Job::loadX3D
    (Ref<Stream::Reader> reader, string contentType,
     Uri sourceUri, int sourceLine, Logger *logger)
    {
      Ref<Server> server;
      try { server = loader->server.getRef(); }
      catch(ForwardDestroyedException &) { return 0; }
      bool progressive = server->getProgressiveLoad();
      Ref<ExecutionContext> context = new ExecutionContext(server, sourceUri
          );
      Ref<Group> rootGroup = new Group(context);
      XML::parse(context, rootGroup, reader, progressive);
      context->setRootGroup(rootGroup);
      return new X3DContents(context);
    }
```

```
Ref<const Loader::Contents> Loader::Job::loadImage
(Ref<Stream::Reader> reader, string contentType,
 Uri sourceUri, Logger *logger)
{
  return new ImageContents(Image(reader, sourceUri.getFile(), Image::
       format_auto, logger));
}

Ref<const Loader::Contents> Loader::Job::loadMovie
(Ref<Stream::Reader> reader, string contentType,
 Uri sourceUri, Logger *logger)
{
  return new MovieContents;
}

Ref<const Loader::Contents> Loader::Job::loadScript
(Ref<Stream::Reader> reader, string contentType,
 Uri sourceUri, int sourceLine, Logger *logger)
{
  string s = Stream::readIntoString(reader);
  return new ScriptContents("ecmascript", string(s), sourceUri.getFile
       (), sourceLine);
}

Ref<const Loader::Contents> Loader::Job::load
(Ref<Stream::Reader> reader, string contentType,
 Uri sourceUri, int sourceLine, Logger *logger)
{
  switch(uriType)
  {
  case uriType_X3D:
    return loadX3D(reader, contentType, sourceUri, sourceLine, logger);
  case uriType_Image:
    return loadImage(reader, contentType, sourceUri, logger);
  case uriType_Movie:
    return loadMovie(reader, contentType, sourceUri, logger);
  case uriType_Script:
    return loadScript(reader, contentType, sourceUri, sourceLine, logger
        );
  default:
    ARCHON_THROW1(InternalException,
                  "X3D::Loader::Job: Unexpected Uri type");
  }
}

/**
 * \todo Problems with inlined ECMAScript - we need both a referer
 * URI and a line number. For the referer it would be appropriate
 * with a null URI, say a URI with scheme = "".
 */
void Loader::Job::main()
{
  for(unsigned i=0; i<uriList.size(); ++i)
  {
    string s = uriList[i];

    /*
```



```
     * Intercept the "ecmascript:" / "javascript:" (deprecated) URI
     * schemes.
     * Note that these inlined schemes are non-cachable.
     */
    const string::size_type p = s.find(':');
    if(p != string::npos)
    {
      string scheme(s, 0, p);
      if(Text::compareIgnoreCase(scheme, "ecmascript") == 0 ||
         Text::compareIgnoreCase(scheme, "javascript") == 0)
      {
        static unsigned long counter = 0;

        PrefixLogger
          logger("[inlined ECMAScript]" +
                 string(counter ? "[" + Text::toString(counter) +
                        "]" : "") + ": ", loader->logger);
        ++counter;

        try
        {
          Ref<Stream::Reader> reader = Stream::makeStringReader(s.substr(
               p+1));
          Ref<const Contents> c =
            load(reader, "text/ecmascript", Uri(), -1, &logger);
          reader.reset();

          Mutex::Lock l(loader->mutex);
          contents = c;
          break;
        }
        catch(Utilities::InternalException &)
        {
          throw;
        }
        catch(ThreadTerminatedException &)
        {
          logger.log("Loading canceled");
          break;
        }
        catch(Image::UnknownFormatException &)
        {
          logger.log("Unknown image format");
        }
        catch(...)
        {
          logger.log("Loading failed");
        }
      }
      continue;
    }


    Uri uri;

    try
    {
      uri = Uri(s, baseUri);
```

```
    }
    catch(Uri::SyntaxException &e)
    {
      loader->logger->log("Malformed URI:\n" + Uri::explain(e));
      continue;
    }

    string absoluteUriString = uri.toString();
    bool cachable = uriType != uriType_X3D;

    // Consult the cache
    if(cachable)
    {
      if(uri.isFileScheme() && fileServer)
        absoluteUriString = "archon://" +
          Text::toString(fileServer->getFileServerId()) +
          uri.getPath();

      Mutex::Lock l(loader->mutex);
      for(;;)
      {
        map<string, Ref<const Contents> >::iterator i =
          loader->cache.find(absoluteUriString);
        if(i == loader->cache.end()) break;
        contents = i->second;
        if(contents) break;
        loader->cacheChange.wait();
      }
      if(contents) break;

      // No hit so allocate a new cache entry
      loader->cache[absoluteUriString];
    }

    string sourceName = uri.getFile();
    int counter = sourceNameCounter(sourceName);
    PrefixLogger
      logger(sourceName + (counter ? "[" + Text::toString(counter) +
                           "]" : "") + ": ", loader->logger);

    bool quit = false;
    try
    {
      Ref<Stream::Reader> reader;
      string contentType;

      if(uri.isFileScheme() && fileServer)
      {
        string path = uri.getPath();
        logger.log("Requesting: " + absoluteUriString);
        reader = fileServer->request(path);
        logger.log("Found");
      }
      else
      {
        logger.log("Requesting: " + absoluteUriString);
        Ref<Web::Client::Response> response =
          webClient->request(Web::Client::Request::get(uri));
```



```
        contentType = response->getContentType();
        logger.log("Found: " + contentType);
        reader = response;
      }
      Ref<const Contents> c = load(reader, contentType, uri, 1, &logger);
      reader.reset();
      logger.log("Loaded");

      {
        Mutex::Lock l(loader->mutex);
        contents = c;
        if(cachable) loader->cache[absoluteUriString] = c;
      }
      if(cachable) loader->cacheChange.notifyAll();
      break;
    }
    catch(Utilities::InternalException &)
    {
      throw;
    }
    catch(ThreadTerminatedException &)
    {
      logger.log("Loading canceled");
      quit = true;
    }
    catch(Image::UnknownFormatException &)
    {
      logger.log("Unknown image format");
    }
    catch(FileServer::RequestException &e)
    {
      logger.log("Loading failed: " + e.getMessage());
    }
    catch(Web::RequestException &e)
    {
      logger.log("Loading failed: " + e.getMessage());
    }
    catch(...)
    {
      logger.log("Loading failed");
    }

    // Remove cache entry for faulty URI
    if(cachable)
    {
      {
        Mutex::Lock l(loader->mutex);
        loader->cache.erase(absoluteUriString);
      }
      loader->cacheChange.notifyAll();
    }
    if(quit) break;
  }

  // Signal completion
  onCompletion();
  completed = true;
  loader->jobCompletion.notifyAll();
```

```
    }


    Loader::Loader(Ref<Server> server, Ref<Web::Client> webClient):
      server(server),
      logger(server->getLogger()),
      webClient(webClient),
      jobQueue(5, Time(30l)),
      cacheChange(mutex), jobCompletion(mutex)
    {
    }

    Loader::~Loader()
    {{ // The extra scope is needed to work around gcc3.2 bug #8287
      wait();
    }}

    void Loader::submitJob(Ref<Job> j)
    {
      jobQueue.add(j.get());
    }
  }
}
```

Listing: **archon/x3d/server/mouse.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_MOUSE_H
#define ARCHON_X3D_MOUSE_H

/*
 * \file mouse.H
 *
 *  X3D Pointing device sensor component
 */

#include <archon/math/vector.H>

#include <archon/x3d/server/core.H>
#include <archon/x3d/server/scene.H>
```



```
namespace Archon
{
  namespace X3D
  {
    using namespace Math;

    struct PointingDeviceSensorNode: SensorNode
    {
      static const NodeType *type;

    protected:
      PointingDeviceSensorNode() {}
    };


    struct DragSensorNode: PointingDeviceSensorNode
    {
      static const NodeType *type;

    protected:
      DragSensorNode():
        autoOffset(true), trackPoint(0, 0, 0) {}
      DragSensorNode(bool autoOffset):
        autoOffset(autoOffset), trackPoint(0, 0, 0) {}

    private:
      friend class Viewer;
      friend void initializeMouseComponent();

      bool autoOffset; // initial value: TRUE
      Vector3 trackPoint;

      EventSource autoOffsetChanged;
      EventSource trackPointChanged;

      Time autoOffsetStamp;
    };


    struct PlaneSensor: DragSensorNode
    {
      static Ref<Node> instantiate(BackRef<ExecutionContext> c)
      {
        return new PlaneSensor(c);
      }

      static const NodeType *type;
      const NodeType *getType() const { return type; }

      PlaneSensor(BackRef<ExecutionContext> c):
        Node(c),
        maxPosition(-1, -1), minPosition(0, 0),
        offset(0, 0, 0), translation(0, 0, 0) {}

    private:
      friend class Viewer;
      friend void initializeMouseComponent();
```

```
  Vector2 maxPosition;
  Vector2 minPosition;
  Vector3 offset;
  Vector3 translation;

  EventSource maxPositionChanged;
  EventSource minPositionChanged;
  EventSource offsetChanged;
  EventSource translationChanged;

  Time maxPositionStamp;
  Time minPositionStamp;
  Time offsetStamp;
};


struct TouchSensorNode: PointingDeviceSensorNode
{
  static const NodeType *type;

protected:
  TouchSensorNode(): isOver(false) {}

private:
  friend class Viewer;
  friend void initializeMouseComponent();

  bool isOver;
  Time touchTime;

  EventSource isOverChanged;
  EventSource touchTimeChanged;
};


struct TouchSensor: TouchSensorNode
{
  static Ref<Node> instantiate(BackRef<ExecutionContext> c)
  {
    return new TouchSensor(c);
  }

  static const NodeType *type;
  const NodeType *getType() const { return type; }

  TouchSensor(BackRef<ExecutionContext> c):
    Node(c),
    hitNormal(0, 0, 0), hitPoint(0, 0, 0),
    hitTexCoord(0, 0) {}

private:
  friend class Viewer;
  friend void initializeMouseComponent();

  Vector3 hitNormal;
  Vector3 hitPoint;
  Vector2 hitTexCoord;
```



```
      EventSource hitNormalChanged;
      EventSource hitPointChanged;
      EventSource hitTexCoordChanged;
    };
  }
}

#endif // ARCHON_X3D_MOUSE_H
```

Listing: **archon/x3d/server/mouse.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

/*
 * \file mouse.C
 *
 *  X3D Pointing device sensor component
 */

#include <archon/x3d/server/field_type.H>
#include <archon/x3d/server/field.H>
#include <archon/x3d/server/mouse.H>

namespace Archon
{
  namespace X3D
  {
    extern long nodeTypeIndexEnumerator;

    const NodeType *PointingDeviceSensorNode::type = 0;
    const NodeType *DragSensorNode::type = 0;
    const NodeType *PlaneSensor::type = 0;
    const NodeType *TouchSensorNode::type = 0;
    const NodeType *TouchSensor::type = 0;

    void initializeMouseComponent()
    {
      vector<const FieldBase *> fields;
```

```
// PointingDeviceSensorNode

PointingDeviceSensorNode::type =
  NodeType::newAbstract("PointingDeviceSensorNode", false, 0,
                        SensorNode::type);


// DragSensorNode

fields.push_back(newExposedField("autoOffset", SFBool::type,
                                 &DragSensorNode::autoOffset,
                                 &DragSensorNode::autoOffsetChanged,
                                 &DragSensorNode::autoOffsetStamp));
fields.push_back(newEventOut("trackPoint", SFVec3f::type,
                             &DragSensorNode::trackPoint,
                             &DragSensorNode::trackPointChanged));
DragSensorNode::type =
  NodeType::newAbstract("DragSensorNode", false, &fields,
                        PointingDeviceSensorNode::type);

fields.clear();


// PlaneSensor

fields.push_back(newExposedField("maxPosition", SFVec2f::type,
                                 &PlaneSensor::maxPosition,
                                 &PlaneSensor::maxPositionChanged,
                                 &PlaneSensor::maxPositionStamp));
fields.push_back(newExposedField("minPosition", SFVec2f::type,
                                 &PlaneSensor::minPosition,
                                 &PlaneSensor::minPositionChanged,
                                 &PlaneSensor::minPositionStamp));
fields.push_back(newExposedField("offset", SFVec3f::type,
                                 &PlaneSensor::offset,
                                 &PlaneSensor::offsetChanged,
                                 &PlaneSensor::offsetStamp));
fields.push_back(newEventOut("translation", SFVec3f::type,
                             &PlaneSensor::translation,
                             &PlaneSensor::translationChanged));
PlaneSensor::type =
  NodeType::newConcrete("PlaneSensor", PlaneSensor::instantiate, &
       fields,
                        DragSensorNode::type);

fields.clear();


// TouchSensorNode

fields.push_back(newEventOut("isOver", SFBool::type,
                             &TouchSensorNode::isOver,
                             &TouchSensorNode::isOverChanged));
fields.push_back(newEventOut("touchTime", SFTime::type,
                             &TouchSensorNode::touchTime,
                             &TouchSensorNode::touchTimeChanged));
TouchSensorNode::type =
  NodeType::newAbstract("TouchSensorNode", false, &fields,
```



```
                        PointingDeviceSensorNode::type);

      fields.clear();


      // TouchSensor

      fields.push_back(newEventOut("hitNormal", SFVec3f::type,
                                   &TouchSensor::hitNormal,
                                   &TouchSensor::hitNormalChanged));
      fields.push_back(newEventOut("hitPoint", SFVec3f::type,
                                   &TouchSensor::hitPoint,
                                   &TouchSensor::hitPointChanged));
      fields.push_back(newEventOut("hitTexCoord", SFVec2f::type,
                                   &TouchSensor::hitTexCoord,
                                   &TouchSensor::hitTexCoordChanged));
      TouchSensor::type =
        NodeType::newConcrete("TouchSensor", TouchSensor::instantiate, &
             fields,
                              TouchSensorNode::type);

      fields.clear();
    }
  }
}
```

Listing: **archon/x3d/server/navigate.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_NAVIGATE_H
#define ARCHON_X3D_NAVIGATE_H

/*
 * \file navigate.H
 *
 *  X3D Navigation component
 */

#include <archon/x3d/server/appear.H>
```

```
namespace Archon
{
  namespace X3D
  {
    struct Viewpoint: BindableNode
    {
      static Ref<Node> instantiate(BackRef<ExecutionContext> c)
      {
        return new Viewpoint(c);
      }

      static const NodeType *type;
      const NodeType *getType() const { return type; }

      Viewpoint(BackRef<ExecutionContext> c):
        Node(c),
        jump(true),
        position(Vector3(0, 0, 10)),
        orientation(AxisRot3(Vector3(0, 0, 1), 0)),
        fieldOfView(M_PI_4),
        centerOfRotation(Vector3::zero()) {}

      bool getJump()                      const { return jump; }
      const Vector3 &getPosition()        const { return position; }
      const AxisRot3 &getOrientation()    const { return orientation; }
      double getFieldOfView()             const { return fieldOfView; }
      const Vector3 &getCenterOfRotation() const { return centerOfRotation; }
      string getDescription()             const { return description; }

      friend void initializeNavigateComponent();

      bool    jump;
      Vector3 position;
      AxisRot3 orientation;
      double  fieldOfView; // 0 - M_PI
      Vector3 centerOfRotation; // Trackball center
      string  description;

      EventSource jumpChanged;
      EventSource positionChanged;
      EventSource orientationChanged;
      EventSource fieldOfViewChanged;
      EventSource centerOfRotationChanged;

      Time jumpStamp;
      Time positionStamp;
      Time orientationStamp;
      Time fieldOfViewStamp;
      Time centerOfRotationStamp;
    };
  }
}

#endif // ARCHON_X3D_NAVIGATE_H
```

Listing: **archon/x3d/server/navigate.C**

```
/*
```



```
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

/*
 * \file navigate.C
 *
 *  X3D Navigation component
 */

#include <archon/x3d/server/field_type.H>
#include <archon/x3d/server/field.H>
#include <archon/x3d/server/navigate.H>

namespace Archon
{
  namespace X3D
  {
    extern long nodeTypeIndexEnumerator;

    const NodeType *Viewpoint::type = 0;

    void initializeNavigateComponent()
    {
      vector<const FieldBase *> fields;

      // Viewpoint

      fields.push_back(newExposedField("jump", SFBool::type,
                                       &Viewpoint::jump,
                                       &Viewpoint::jumpChanged,
                                       &Viewpoint::jumpStamp));
      fields.push_back(newExposedField("position", SFVec3f::type,
                                       &Viewpoint::position,
                                       &Viewpoint::positionChanged,
                                       &Viewpoint::positionStamp));
      fields.push_back(newExposedField("orientation", SFRotation::type,
                                       &Viewpoint::orientation,
                                       &Viewpoint::orientationChanged,
                                       &Viewpoint::orientationStamp));
      fields.push_back(newExposedField("fieldOfView", SFFloat::type,
                                       &Viewpoint::fieldOfView,
                                       &Viewpoint::fieldOfViewChanged,
                                       &Viewpoint::fieldOfViewStamp));
```

```
      fields.push_back(newExposedField("centerOfRotation", SFVec3f::type,
                                       &Viewpoint::centerOfRotation,
                                       &Viewpoint::centerOfRotationChanged,
                                       &Viewpoint::centerOfRotationStamp));
      fields.push_back(newPrivateField("description", SFString::type,
                                       &Viewpoint::description));
      Viewpoint::type =
        NodeType::newConcrete("Viewpoint", Viewpoint::instantiate, &fields,
                              BindableNode::type);
      fields.clear();
    }
  }
}
```

Listing: **archon/x3d/server/network.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_NETWORK_H
#define ARCHON_X3D_NETWORK_H

/**
 * \file network.H
 *
 * X3D Network component
 */

#include <vector>

#include <archon/x3d/server/scene.H>

namespace Archon
{
  namespace X3D
  {
    /**
     * The base class for all X3D nodes with a single URI field such
     * as ImageTexture and Inline.
     *
     * Asynchronious download of URI contents is automatically
     * initiated (and aborted) as a result of assignment to the URI
```



```
     * field. Descendants of this class may choose to override the
     * 'clear' method to get a notification of when the URI contents
     * is deleted as a consequence of assignment to the URI field.
     *
     * \sa UrlObject::getContents
     * \sa UrlObject::clear
     *
     * \todo It would proberbly be a good idea to utilize the new
     * RefObjectBase::dispose method instead of the rather complex use
     * of Utilities::Ref<T>::getSafeRef.
     */
    struct UrlObject: virtual Node
    {
      static const NodeType *type;

      /**
       * Returns the currently loaded URI contents if any.
       *
       * This method will return the null reference if either the URI
       * field is empty or the download is not yet complete or the
       * download has failed.
       */
      Ref<const Loader::Contents> getContents();

    protected:
      UrlObject();
      virtual ~UrlObject();

      /**
       * Does not excite 'changed' events. Contents loading is dealt
       * with though.
       */
      void addUri(string, Time stamp);

      /**
       * Override this method if you want a chance to clean up any
       * data produced from old URI contents such as OpenGL textures
       * and script engines when the URI is reassigned.
       *
       * Will be called with a scenegraph write lock or a Node
       * lock.
       */
      virtual void clear() {}

      /**
       * Override this method to get a notification of when the URI
       * contents is loaded.
       *
       * BEWARE though, that this method is called asynchroniously by
       * the loader thread. this means that you generally cannot
       * access anything which is changed by other threads without
       * locking a mutex. This includes everything which may be
       * updated during an event cascade (all X3D fields).
       *
       * You should not do too much work from this method since the
       * UrlObject may get orphaned at any time, meaning that the only
       * reference to it will be the one held by the loader
       * thread. This means that the performed work will be in
```

```
     * vain. You may try to detect the orphaned condition with
     * 'RefObjectBase::use_count() == 1'.
     *
     * \note Neither the scenegraph nor the node is locked when this
     * method is called.
     */
    virtual void onLoaded() {}

    /**
     * This method is called whenever a new value is assigned to the
     * 'url' filed to determine if the loading should start
     * immediatly.
     *
     * The default is that loading will start immediately.
     *
     * You may override this method if you want to change the
     * default.
     *
     * \note This method is called with either a scenegraph write
     * lock or a Node lock.
     */
    virtual bool mayStartLoading() const { return true; }

    /**
     * If loading of the contents corresponding to the current value
     * of the 'url' field is not yet startet, then start it now.
     *
     * \note You must always call this method with either a
     * scenegraph write lock or a Node lock.
     *
     * \note This method is automatically called when a new value is
     * assigned to the 'url' field if 'mayStartLoading' returns
     * true.
     */
    void startLoading();

  private:
    friend void initializeNetworkComponent();

    vector<string> url;

    EventSource urlChanged;

    Time urlStamp;


    /**
     * Apart from being a Loader::Job this class is responsible for
     * communicating with the UrlObject which created it. It must be
     * capable of two things: To die on command which is also a
     * promise to not contact the UrlObject again and it must be
     * able to push the loaded contents into the UrlObject.
     *
     * Because we requre that the LoaderJob be able to push contents
     * to the UrlObject it must have a reference the UrlObject. On
     * the other hand we do not want the loader job give the
     * UrlObject unnatural long life because it holds the last
     * reference count for the UrlObject. That is, we do not want
```



```
     * the work of the loader to be in vain. This forces us to use a
     * conventional pointer in the loader job instead of default
     * choise Ref<LoaderJob>.
     *
     * Using a conventional pointer in the loader thread as a
     * reference to the UrlObject imposes another problem: Is the
     * pointer still valid or is the UrlObject deleted (dangeling
     * pointer)? The sollution to this problem is a strict contract
     * between the loader job and the UrlObject saying that the
     * UrlObject must always call the 'kill' method of this class
     * before returning from the destructor. The 'kill' method will
     * then nullify the UrlObject pointer (and send a termination
     * request to the loader thread).
     *
     * From the point of view of the UrlObject a return from
     * 'LoaderJob::kill' is a guarantee that that loader job will
     * never try to contact the UrlObject again. For the UrlObject
     * this guarantee is needed when the value of the URI field
     * changes. In this case the UrlObject wants to start a new
     * loader job, and without care the old job could complete later
     * than the new job and therby replace the new contents with the
     * old. Another reason to call 'LoaderJob::kill' when the URI
     * field changes is that any further work carried out by the old
     * job will be in vain.
     *
     * Apart from som details on the use of mutexes this should be
     * the whole story about the rather delicate communication
     * between the UrlObject and its loader job(s).
     *
     * \sa Utilities::Ref<T>::getSafeRef
     * \sa kill
     * \sa onCompletion
     * \sa setContents
     * \sa urlAssigned
     */
    struct LoaderJob: Loader::Job
    {
      LoaderJob(UrlObject *urlObject, Ref<Loader> loader, Uri baseUri,
                const vector<string> &uriList, Loader::UriType uriType,
                Ref<Loader::FileServer> fileServer):
        Loader::Job(loader, baseUri, uriList, uriType, fileServer),
        urlObject(urlObject) {}

      void kill();

    private:
      void onCompletion();

      Mutex mutex;
      UrlObject *urlObject;
    };


    /**
     * Protects against simultanious access to 'loaderJob' and
     * 'contents' from the loader thread and the event handling
     * thread.
     */
```

```
    Mutex loaderMutex;

    Ref<LoaderJob> loaderJob;
    Ref<const Loader::Contents> contents;

    void setContents(Ref<const Loader::Contents> c);

    void killLoaderJob();

    /**
     * Must be called with a scenegraph write lock or a Node
     * lock. Is currently called during event cascading/assigning
     * only.
     */
    void urlAssigned(const vector<string> &);

    virtual Loader::UriType getUriType() const = 0;
  };

  /**
   * Base class for nodes which can contain sub contexts.
   *
   * \todo We proberbly neeed to add a timestamp for the
   * subContextVariable to support Node::changedSince. Also a
   * timestamp is needed in the rootGroup variable of
   * ExecutionContext.
   */
  struct SubContextNode: virtual ChildNode
  {
    static const NodeType *type;

    Ref<ExecutionContext> getSubContext() const;
    void setSubContext(Ref<ExecutionContext>);

    /**
     * Requires scenegraph read lock.
     */
    Ref<ExecutionContext> getSubContextNoLock() const { return subContext
         ; }

  protected:
    SubContextNode() {}

    Ref<ExecutionContext> subContext;

    void refForwardDestroy();
  };

  struct Inline: SubContextNode, UrlObject
  {
    static Ref<Node> instantiate(BackRef<ExecutionContext> c)
    {
      return new Inline(c);
    }

    static const NodeType *type;
    const NodeType *getType() const { return type; }
```



```
    Inline(BackRef<ExecutionContext> c): Node(c) {}

  private:
    friend void initializeNetworkComponent();

    bool load;

    EventSource loadChanged;

    Time loadStamp;

    Loader::UriType getUriType() const
    {
      return Loader::uriType_X3D;
    }

    /**
     * Must be called with a scenegraph write lock or a Node
     * lock. Is currently called during event cascading/assigning
     * only.
     */
    void loadAssigned(const bool &);

    void onLoaded(); // Overriding from UrlObject
    void clear(); // Overriding from UrlObject
  };

  struct ApplicationLink: SubContextNode
  {
    static Ref<Node> instantiate(BackRef<ExecutionContext> c)
    {
      return new ApplicationLink(c);
    }

    static const NodeType *type;
    const NodeType *getType() const { return type; }

    ApplicationLink(BackRef<ExecutionContext> c): Node(c) {}

  private:
    friend void initializeNetworkComponent();

    string dump(int level) const;
  };
 }
}

#endif // ARCHON_X3D_NETWORK_H
```

Listing: **archon/x3d/server/network.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
```

```
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

/**
 * \file network.C
 *
 * X3D Network component
 */

#include <archon/util/text.H>

#include <archon/x3d/server/field_type.H>
#include <archon/x3d/server/field.H>
#include <archon/x3d/server/network.H>
#include <archon/x3d/server/scene.H>
#include <archon/x3d/server/server.H>

namespace Archon
{
  namespace X3D
  {
    extern long nodeTypeIndexEnumerator;

    const NodeType *UrlObject::type = 0;
    const NodeType *SubContextNode::type = 0;
    const NodeType *Inline::type = 0;
    const NodeType *ApplicationLink::type = 0;

    void initializeNetworkComponent()
    {
      vector<const FieldBase *> fields;

      // UrlObject

      fields.push_back(newExposedField("url", MFString::type,
                                       &UrlObject::url,
                                       &UrlObject::urlAssigned,
                                       &UrlObject::urlChanged,
                                       &UrlObject::urlStamp));
      UrlObject::type =
        NodeType::newAbstract("UrlObject", false, &fields, Node::type);

      fields.clear();


      // SubContextNode

      SubContextNode::type =
        NodeType::newAbstract("SubContextNode", false, 0, ChildNode::type);
```



```
      // Inline

      fields.push_back(newExposedField("load", SFBool::type,
                                       &Inline::load,
                                       &Inline::loadAssigned,
                                       &Inline::loadChanged,
                                       &Inline::loadStamp));
      Inline::type =
        NodeType::newConcrete("Inline", Inline::instantiate,
                              &fields, SubContextNode::type, UrlObject::type
                                   );
      fields.clear();


      // ApplicationLink

      ApplicationLink::type =
        NodeType::newConcrete("ApplicationLink", ApplicationLink::instantiate
             ,
                              0, SubContextNode::type);
      fields.clear();
    }

    UrlObject::UrlObject()
    {
    }

    UrlObject::~UrlObject()
    {{ // The extra scope is needed to work around gcc3.2 bug #8287
      killLoaderJob();
    }}

    Ref<const Loader::Contents> UrlObject::getContents()
    {
      Mutex::Lock l(loaderMutex);
      return contents;
    }

    /**
     * \todo This one should instead utilize the soon coming
     * SimpleSequenceFieldBase::add(). In fact this code may even be
     * used to implement Node::add.
     */
    void UrlObject::addUri(string u, Time stamp)
    {
      const FieldBase *f = getType()->lookupField("url");
      if(!f) ARCHON_THROW1(InternalException,
                           "Could not find field 'url' in UrlObject");
      {
        Mutex::Lock l(mutex);
        if(!isRealizedNoLock())
        {
          url.push_back(u);
          Event e(new SimpleValue<vector<string> >(MFString::type, url),
               stamp);
          f->handleEvent(this, &e, false);
```

```
      return;
    }
  }
  try
  {
    Mutex::Lock l(context->server->scenegraphMutex);
    url.push_back(u);
    Event e(new SimpleValue<vector<string> >(MFString::type, url), stamp
        );
    f->handleEvent(this, &e, true);
  }
  catch(ForwardDestroyedException &) {}
}

/**
 * This method serves as a means for the UrlObject to discard
 * the loader job and to ensure that the loader job, which is
 * an autonomous entity, will not attempt any further access
 * to the UrlObject. This is important for the UrlObject in
 * two situations: When the UrlObject is about to be deleted
 * and when it want to request a new URI as a result of
 * assinment to the URI field of the UrlObject.
 *
 * \sa LoaderJob
 */
void UrlObject::LoaderJob::kill()
{
  Loader::Job::kill(); // Ask this job to abort
  Mutex::Lock l(mutex);
  urlObject = 0;
}

/**
 * This method gets called by the base class 'Loader::Job' when
 * the job is complete. The task for this method is to push the
 * loaded contents to the UrlObject. Note that, by placing the
 * call to 'UrlObject::setContents' in the mutex we guarantee that
 * the access is only fenformed if UrlObject has not already
 * returned from a call to 'LoaderJob::kill'.
 *
 * Note the use of Ref<T>(T *, RefSafeIncTag) for aquiring a
 * reference to the UrlObject. If successful (non-null) it
 * guarantees not only that the UrlObject stays alive, but also
 * that it has not been partially destroyed. That is, degraded
 * from its original concrete type to the abstract type
 * UrlObject. Note that htis is VERY important since
 * UrlObject::onLoaded is a virtual method and thus executes code
 * which assumes to be in the context of the derived type.
 */
void UrlObject::LoaderJob::onCompletion()
{
  Ref<const Loader::Contents> c = getContents();

  Mutex::Lock l(mutex);

  // Attempt to aquire a "safe" reference
  Ref<UrlObject> safeRef(urlObject, RefSafeIncTag());
  if(!safeRef) return;
```



```
  urlObject->setContents(c);

  l.release();

  safeRef->onLoaded();
}

/**
 * This method is exclusivly for the loader job. It is used by
 * the loader job to push loaded contents to the UrlObject. As a
 * side effect it deletes the loader job reference held by the
 * UrlObject, and this ensures that when the loader thread
 * terminates (shortly hereafter) its memory resources will be
 * deleted.
 */
void UrlObject::setContents(Ref<const Loader::Contents> c)
{
  Mutex::Lock l(loaderMutex);
  contents = c;
  loaderJob = 0;
}

/**
 * Prevent further access from the old loader job.
 *
 * Note that if 'LoaderJob::kill' was called within the mutex, it
 * would cause a dead-lock.
 *
 * Also note that because a loader job may exist and may access
 * the 'loaderJob' variable at any time via 'setContents' we must
 * not access the 'loaderJob' variable unless we are inside the
 * mutex.
 */
void UrlObject::killLoaderJob()
{
  Ref<LoaderJob> j;
  {
    Mutex::Lock l(loaderMutex);
    j = loaderJob;
    loaderJob.reset();
    contents.reset();
  }
  if(j) j->kill();
}

void UrlObject::startLoading()
{
  if(!url.size()) return;
  Ref<ExecutionContext> c;
  Ref<Loader> loader;
  Ref<Loader::FileServer> fileServer;
  try
  {
    c = context.getRef();
    loader = c->server->getLoader();
    fileServer = c->fileServer.getRef();
  }
```

```
    catch(ForwardDestroyedException &) { return; }
    if(!loader) return;
    Ref<LoaderJob> j =
      new LoaderJob(this, loader, c->getBaseUri(),
                    url, getUriType(), fileServer);
    {
      Mutex::Lock l(loaderMutex);
      if(contents || loaderJob) return;
      loaderJob = j;
      loader->submitJob(j);
    }
  }

  void UrlObject::urlAssigned(const vector<string> &)
  {
    killLoaderJob();

    /*
     * Give the derived node a chance to clean up any data
     * produced from the old URI contents such as textures and
     * script engines.
     */
    clear();

    if(mayStartLoading()) startLoading();
  }


  Ref<ExecutionContext> SubContextNode::getSubContext() const
  {
    {
      Mutex::Lock l(mutex);
      if(!isRealizedNoLock()) return subContext;
    }
    try
    {
      Mutex::Lock l(context->server->scenegraphMutex);
      return subContext;
    }
    catch(ForwardDestroyedException &) { return 0; }
  }

  void SubContextNode::setSubContext(Ref<ExecutionContext> c)
  {
    if(c) c->realize();
    {
      Mutex::Lock l(mutex);
      if(!isRealizedNoLock())
      {
        subContext = c;
        return;
      }
    }
    try
    {
      Mutex::Lock l(context->server->scenegraphMutex);
      subContext = c;
    }
```



```
      catch(ForwardDestroyedException &) {}
    }

    void SubContextNode::refForwardDestroy()
    {
      Node::refForwardDestroy();
      subContext.reset();
    }


    void Inline::loadAssigned(const bool &)
    {
      if(load) startLoading();
    }

    void Inline::onLoaded()
    {
      Ref<const Loader::Contents> c = getContents();
      if(!c) return;
      const Loader::X3DContents *d =
        dynamic_cast<const Loader::X3DContents *>(c.get());
      if(!d) ARCHON_THROW1(InternalException, "Wrong content type");
      setSubContext(d->context);
    }

    void Inline::clear()
    {
      subContext.reset();
    }


    string ApplicationLink::dump(int level) const
    {
      if(!subContext) return "";
      return subContext->dump(level);
    }
  }
}
```

Listing: **archon/x3d/server/node.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
```

```
 */

#ifndef ARCHON_X3D_NODE_H
#define ARCHON_X3D_NODE_H

/**
 * \file node.H
 *
 * X3D Basic node
 */

#include <string>

#include <archon/util/logger.H>
#include <archon/util/ref.H>

#include <archon/x3d/server/exception.H>
#include <archon/x3d/server/type.H>

namespace Archon
{
  namespace X3D
  {
    struct ExecutionContext; // "scene.H"
    struct Parser;           // "parse.H"
    struct RouteHead;        // "event.H"

    struct NodeNameInUseException: X3D::ArgumentException
    {
      NodeNameInUseException(string l): Utilities::Exception(l) {}
    };

    /**
     * The common base for all X3D node types.
     */
    struct Node: virtual BackRefObjectBase
    {
      static const NodeType *type;
      virtual const NodeType *getType() const = 0;

      struct Exception: virtual X3D::Exception {};
      struct ArgumentException: Exception, X3D::ArgumentException
      {
        ArgumentException(string l, string m): Utilities::Exception(l, m) {}
      };
      struct StateException: Exception, X3D::StateException
      {
        StateException() {}
        StateException(string l, string m): Utilities::Exception(l, m) {}
      };
      struct InternalException: Exception, X3D::InternalException
      {
        InternalException(string l, string m): Utilities::Exception(l, m) {}
      };
      /**
       * Should be thrown by methods that are illegal on realized
       * nodes.
       */
```



```
      struct RealizedException: StateException
      {
        RealizedException(string l): Utilities::Exception(l) {}
      };
      /**
       * Should be thrown when the combination of the phase of the
       * nodes lifecycle and the input/output capabilities of the
       * field is not compatible with the requested operation.
       */
      struct AccessException: StateException
      {
        AccessException(string l): Utilities::Exception(l) {}
      };
      /**
       * Should be thrown when a field of sequence type is shorter
       * that a specified index.
       */
      struct RangeException: StateException
      {
        RangeException(string l): Utilities::Exception(l) {}
      };

      /**
       * If this node is not already realized, then realize it now.
       *
       * A node must be realized before it receives any event and
       * before it is being rendered.
       *
       * Must not be called with a scenegraph lock.
       */
      void realize();

      bool isRealized() const;

      /**
       * Will never be called with a scenegraph lock or a lock on this
       * node.
       */
      virtual void onRealized() {}

      string getName() const;

      void setName(string name)
        throw(NodeNameInUseException);

      /**
       * Search for a field of the specified name in this node.
       * 'set_*' and '*_changed' variants are tryed automatically.
       *
       * Note that the default is just to consult the associated node
       * type. This may be overwritten by nodes that can contain
       * custom fields such as Script nodes. BUT BE SURE TO MAKE IT
       * REENTRANT.
       *
       * \return 0 if not found.
       *
       * \sa NodeType::lookupField
       * \sa Script::lookupField
```

```
  */
 virtual const FieldBase *lookupField(string name) const;

 /**
  * Must not be called with a scenegraph lock.
  */
 Ref<ValueBase> get(const FieldBase *) const throw(AccessException);

 /**
  * Must not be called with a scenegraph lock.
  */
 unsigned getSize(const SequenceFieldBase *) const
   throw(AccessException);

 /**
  * Must not be called with a scenegraph lock.
  */
 Ref<ValueBase> getAt(const SequenceFieldBase *, unsigned index) const
   throw(AccessException, RangeException);

 /**
  * Must not be called with a scenegraph lock.
  *
  * \todo Verify asyclic property
  */
 void handleEvent(const FieldBase *, const Event *)
   throw(AccessException);

 /**
  * Must not be called with a scenegraph lock.
  *
  * \todo Verify asyclic property
  */
 void setAt(const SequenceFieldBase *, Ref<const ValueBase>,
            unsigned index, Time) throw(AccessException, RangeException
                 );

 /**
  * Must not be called with a scenegraph lock.
  *
  * \todo Verify asyclic property
  */
 void add(const SequenceFieldBase *, Ref<const ValueBase>, Time)
   throw(AccessException);

 /**
  * Must not be called with a scenegraph lock.
  *
  * \todo Verify asyclic property
  */
 void remove(const SequenceFieldBase *, Ref<const ValueBase>, Time)
   throw(AccessException);

 /**
  * Convenience wrapper for add(const SequenceFieldBase *,
  * Ref<ValueBase>, Time)
  */
 void add(const NodeSequenceFieldBase *, Ref<Node>, Time)
```



```
   throw(AccessException);

 /**
  * Convenience wrapper for remove(const SequenceFieldBase *,
  * Ref<ValueBase>, Time)
  */
 void remove(const NodeSequenceFieldBase *, Ref<Node>, Time)
   throw(AccessException);

 /**
  * Must not be called with a scenegraph lock.
  *
  * \todo Verify asyclic property
  */
 bool inject(const NodeFieldBase *, Ref<Node> childNode, Time stamp)
   throw(AccessException);

 /**
  * Is any field of this node changed?
  *
  * Must be called with a scenegraph read lock.
  */
 virtual bool changedSince(const Time &t) const;

 /**
  * Requires scenegraph read lock.
  */
 virtual string dump(int level) const;

 /**
  * The executionContext that this node bolongs to.
  *
  * Note: It is importatnt that this association can never
  * change, since this anables safe locking on the scene mutex
  * throug a node. In case you want to transfer a node from one
  * scene to another, you have to clone it and then delete the
  * old one.
  */
 const BackRef<ExecutionContext> context;

protected:
 /**
  * \param context The ExecutionContext that this node
  * belongs to. Must not be null.
  */
 Node(BackRef<ExecutionContext> context);

 /**
  * This constructor must never be called, and if it does, it
  * throws. It is here because of notational convenience. That
  * is, if it wasn't here, then Node would not have a default
  * constructor. This in turn means that every abstract sub-class
  * such as ChildNode must include an explicit construction of
  * Node with a executionContext argument. This is completely
  * redundant ans therefore inconvenient since Node is
  * required to be a virtual base class of all node types, and
  * therefor an abstract node type will never get to call any
  * Node constructor itself.
```

```
     */
    Node();

    virtual ~Node();

    virtual void refForwardDestroy();

    bool isRealizedNoLock() const { return realized; }

    const Mutex mutex;

  private:
    friend struct Route;
    Mutex routeTargetsMutex;
    list<Ref<const RouteHead> > routeTargets;
    void addRouteTarget(Ref<const RouteHead>);
    void removeRouteTarget(Ref<const RouteHead>);

    /**
     * The name of the node.
     *
     * An empty string indicates that this is an un-named node, and
     * therefore it cannot be referred to by name (eg. with a VRML
     * USE statement).
     */
    string name;

    bool realized;
  };
 }
}

#endif // ARCHON_X3D_NODE_H
```

Listing: **archon/x3d/server/node.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

/**
 * \file node.C
 *
```



```
 * X3D Basic node
 */

#include <archon/x3d/server/field_type.H>
#include <archon/x3d/server/node.H>
#include <archon/x3d/server/scene.H>
#include <archon/x3d/server/server.H>

namespace Archon
{
  namespace X3D
  {
    const NodeType *Node::type = 0;

    Node::Node(BackRef<ExecutionContext> context):
      context(context), realized(false)
    {
      if(!context)
        ARCHON_THROW1(ArgumentException,
                      "ExecutionContext required to construct node");
    }

    Node::Node()
    {
      ARCHON_THROW1(InternalException,
                    "Construction of abstract node type");
    }

    Node::~Node()
    {
      try { context->changeNodeName(this, name, ""); } // Erase the mapping
      catch(ForwardDestroyedException &) {}
    }

    void Node::refForwardDestroy()
    {
      getType()->forwardClear(this);
      routeTargets.clear();
    }

    void Node::realize()
    {
      {
        Mutex::Lock l(mutex);
        if(realized) return;
        realized = true;
      }
      onRealized();
    }

    bool Node::isRealized() const
    {
      Mutex::Lock l(mutex);
      return realized;
    }

    string Node::getName() const
    {
```

```
  Mutex::Lock l(mutex);
  return name;
}

void Node::setName(string newName)
  throw(NodeNameInUseException)
{
  Mutex::Lock l(mutex);
  context->changeNodeName(this, name, newName);
  name = newName;
}

const FieldBase *Node::lookupField(string name) const
{
  return getType()->lookupField(name);
}

Ref<ValueBase> Node::get(const FieldBase *f) const
  throw(AccessException)
{
  {
    Mutex::Lock l(mutex);
    if(!realized)
    {
      if(f->getIsEventTarget() != f->getIsEventSource())
        ARCHON_THROW(AccessException);
      return f->get(this);
    }
  }
  try
  {
    if(!f->getIsEventSource()) ARCHON_THROW(AccessException);
    Mutex::Lock l(context->server->scenegraphMutex);
    return f->get(this);
  }
  catch(ForwardDestroyedException &)
  {
    ARCHON_THROW(AccessException);
  }
}

unsigned Node::getSize(const SequenceFieldBase *f) const
  throw(AccessException)
{
  {
    Mutex::Lock l(mutex);
    if(!realized)
    {
      if(f->getIsEventTarget() != f->getIsEventSource())
        ARCHON_THROW(AccessException);
      return f->getSize(this);
    }
  }
  try
  {
    if(!f->getIsEventSource()) ARCHON_THROW(AccessException);
    Mutex::Lock l(context->server->scenegraphMutex);
    return f->getSize(this);
```



```
  }
  catch(ForwardDestroyedException &)
  {
    ARCHON_THROW(AccessException);
  }
}

Ref<ValueBase> Node::getAt(const SequenceFieldBase *f, unsigned i) const
  throw(AccessException, RangeException)
{
  {
    Mutex::Lock l(mutex);
    if(!realized)
    {
      if(f->getIsEventTarget() != f->getIsEventSource())
        ARCHON_THROW(AccessException);
      if(i >= f->getSize(this)) ARCHON_THROW(RangeException);
      return f->getAt(this, i);
    }
  }
  try
  {
    if(!f->getIsEventSource()) ARCHON_THROW(AccessException);
    Mutex::Lock l(context->server->scenegraphMutex);
    if(i >= f->getSize(this)) ARCHON_THROW(RangeException);
    return f->getAt(this, i);
  }
  catch(ForwardDestroyedException &)
  {
    ARCHON_THROW(AccessException);
  }
}

/**
 * \todo Add value type verification.
 */
void Node::handleEvent(const FieldBase *f, const Event *e)
  throw(AccessException)
{
  // If the event value consists of one or more nodes then verify
  // and realize them
  if(const NodeValue *v = dynamic_cast<const NodeValue *>(e->value.get
      ()))
  {
    if(v->value)
    {
      if(v->value->context != context)
        ARCHON_THROW1(ArgumentException,
                      "handleEvent: Node has wrong context");
      v->value->realize();
    }
  }
  else if(const NodeSequenceValue *v =
          dynamic_cast<const NodeSequenceValue *>(e->value.get()))
    for(unsigned i=0; i<v->value.size(); ++i)
    {
      if(!v->value[i])
        ARCHON_THROW1(ArgumentException,
```

```
                  "handleEvent: null found in node vector");
      if(v->value[i]->context != context)
        ARCHON_THROW1(ArgumentException,
                  "handleEvent: Node has wrong context");
      v->value[i]->realize();
    }

  {
    Mutex::Lock l(mutex);
    if(!realized)
    {
      if(f->getIsEventTarget() != f->getIsEventSource())
        ARCHON_THROW(AccessException);
      f->handleEvent(this, e, false);
      return;
    }
  }
  try
  {
    if(!f->getIsEventTarget()) ARCHON_THROW(AccessException);
    Mutex::Lock l(context->server->scenegraphMutex);
    f->handleEvent(this, e, true);
  }
  catch(ForwardDestroyedException &)
  {
    ARCHON_THROW(AccessException);
  }
}

/**
 * \todo Add value type verification.
 */
void Node::setAt(const SequenceFieldBase *f, Ref<const ValueBase> v,
                 unsigned i, Time t) throw(AccessException,
                      RangeException)
{
  if(const NodeValue *w = dynamic_cast<const NodeValue *>(v.get()))
  {
    if(!w->value)
      ARCHON_THROW1(ArgumentException, "setAt: Node was null");
    if(w->value->context != context)
      ARCHON_THROW1(ArgumentException, "setAt: Node has wrong context");
    w->value->realize();
  }

  {
    Mutex::Lock l(mutex);
    if(!realized)
    {
      if(f->getIsEventTarget() != f->getIsEventSource())
        ARCHON_THROW(AccessException);
      if(i >= f->getSize(this)) ARCHON_THROW(RangeException);
      f->setAt(this, v.get(), i, t, false);
      return;
    }
  }
  try
  {
```



```
    if(!f->getIsEventTarget() || !f->getIsEventSource())
      ARCHON_THROW(AccessException);
    Mutex::Lock l(context->server->scenegraphMutex);
    if(i >= f->getSize(this)) ARCHON_THROW(RangeException);
    f->setAt(this, v.get(), i, t, true);
  }
  catch(ForwardDestroyedException &)
  {
    ARCHON_THROW(AccessException);
  }
}

/**
 * \todo Add value type verification.
 */
void Node::add(const SequenceFieldBase *f, Ref<const ValueBase> v, Time t
    )
  throw(AccessException)
{
  if(const NodeValue *w = dynamic_cast<const NodeValue *>(v.get()))
  {
    if(!w->value)
      ARCHON_THROW1(ArgumentException, "add: Node was null");
    if(w->value->context != context)
      ARCHON_THROW1(ArgumentException, "add: Node has wrong context");
    w->value->realize();
  }

  {
    Mutex::Lock l(mutex);
    if(!realized)
    {
      if(f->getIsEventTarget() != f->getIsEventSource())
        ARCHON_THROW(AccessException);
      f->add(this, v.get(), t, false);
      return;
    }
  }
  try
  {
    if(!f->getIsEventTarget() || !f->getIsEventSource())
      ARCHON_THROW(AccessException);
    Mutex::Lock l(context->server->scenegraphMutex);
    f->add(this, v.get(), t, true);
  }
  catch(ForwardDestroyedException &)
  {
    ARCHON_THROW(AccessException);
  }
}

/**
 * \todo Add value type verification.
 */
void Node::remove(const SequenceFieldBase *f, Ref<const ValueBase> v,
                  Time t) throw(AccessException)
{
  if(const NodeValue *w = dynamic_cast<const NodeValue *>(v.get()))
```

```
    {
      if(!w->value)
        ARCHON_THROW1(ArgumentException, "remove: Node was null");
      if(w->value->context != context)
        ARCHON_THROW1(ArgumentException, "remove: Node has wrong context");
      w->value->realize();
    }

    {
      Mutex::Lock l(mutex);
      if(!realized)
      {
        if(f->getIsEventTarget() != f->getIsEventSource())
          ARCHON_THROW(AccessException);
        f->remove(this, v.get(), t, false);
        return;
      }
    }
    try
    {
      if(!f->getIsEventTarget() || !f->getIsEventSource())
        ARCHON_THROW(AccessException);
      Mutex::Lock l(context->server->scenegraphMutex);
      f->remove(this, v.get(), t, true);
    }
    catch(ForwardDestroyedException &)
    {
      ARCHON_THROW(AccessException);
    }
  }

  void Node::add(const NodeSequenceFieldBase *f, Ref<Node> n, Time t)
    throw(AccessException)
  {
    add(f, new NodeValue(SFNode::type, n.get()), t);
  };

  void Node::remove(const NodeSequenceFieldBase *f, Ref<Node> n, Time t)
    throw(AccessException)
  {
    remove(f, new NodeValue(SFNode::type, n.get()), t);
  };


  bool Node::inject(const NodeFieldBase *f, Ref<Node> n, Time stamp)
    throw(AccessException)
  {
    if(!n) ARCHON_THROW1(ArgumentException,
                         "inject: Node was null");
    if(n->context != context)
      ARCHON_THROW1(ArgumentException,
                    "inject: Node has wrong context");

    n->realize();

    {
      Mutex::Lock l(mutex);
      if(!realized)
```



```
      {
        if(f->getIsEventTarget() != f->getIsEventSource())
          ARCHON_THROW(AccessException);
        return f->inject(this, n.get(), stamp, false);
      }
    }
    try
    {
      if(!f->getIsEventTarget()) ARCHON_THROW(AccessException);
      Mutex::Lock l(context->server->scenegraphMutex);
      return f->inject(this, n.get(), stamp, true);
    }
    catch(ForwardDestroyedException &)
    {
      ARCHON_THROW(AccessException);
    }
  };

  bool Node::changedSince(const Time &t) const
  {
    return getType()->changedSince(this, t);
  }

  string Node::dump(int level) const
  {
    return getType()->dumpNode(this, level);
  }

  void Node::addRouteTarget(Ref<const RouteHead> r)
  {
    Mutex::Lock l(routeTargetsMutex);
    routeTargets.push_back(r);
  }

  void Node::removeRouteTarget(Ref<const RouteHead> r)
  {
    Mutex::Lock l(routeTargetsMutex);
    routeTargets.remove(r);
  }


  void initializeNode()
  {
    Node::type = NodeType::newAbstract("Node", true, 0);
  }
 }
}
```

Listing: **archon/x3d/server/parse.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
```

```
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_PARSE_H
#define ARCHON_X3D_PARSE_H

#include <string>

#include <archon/util/ref.H>
#include <archon/util/logger.H>

#include <archon/x3d/server/core.H>
#include <archon/x3d/server/group.H>
#include <archon/x3d/server/scene.H>

namespace Archon
{
  namespace X3D
  {
    using namespace std;
    using namespace Utilities;

    struct CustomFieldNode; // Defined in "custom_field.H"
    struct CustomFieldBase; // Defined in "custom_field.H"

    struct ProgressTracker
    {
      virtual void set(Ref<Scene>) = 0;
      virtual void progress(long chars_loaded) = 0;
      virtual void scene(const string &uri) = 0;
      virtual void image(const string &uri) = 0;
    };

    struct Parser
    {
      struct Exception: virtual X3D::Exception {};
      struct ResourceException: Exception, X3D::ResourceException
      {
        ResourceException() {}
        ResourceException(string l, string m): Utilities::Exception(l, m) {}
      };
      struct ParseException: ResourceException
      {
        ParseException(string l, string m): Utilities::Exception(l, m) {}
      };

    protected:
      virtual void warning(string message) { logger->log(message); }
```



```
      Parser(Ref<ExecutionContext> context, Ref<GroupingNode> rootGroup,
             string sourceName, Logger *logger):
        context(context), rootGroup(rootGroup), sourceName(sourceName),
             logger(logger) {}

      virtual ~Parser() {}

      Ref<ExecutionContext> context;

      /**
       * The root group will normally be the root group of the scene,
       * but may also be some other group if a new file is parsed in
       * the context of an existing scene.
       */
      Ref<GroupingNode> rootGroup;

      /**
       * Name of source for parsed data
       */
      string sourceName;

      Logger *logger;

      bool validateIdentifier(string);
      void addNamedNode(string, Ref<Node>);

      Ref<Node> lookupNode(string name) { return context->lookupNode(name); }

      void addRootGroupChild(Ref<ChildNode>, Time stamp);

      void addRoute(string sourceNodeName, string sourceFieldName,
                    string targetNodeName, string targetFieldName);

      int getNextCustomFieldIndex(Ref<CustomFieldNode>);
      void addScriptField(Ref<CustomFieldNode>, Ref<CustomFieldBase>);
    };
  }
}

#endif // ARCHON_X3D_PARSE_H
```

Listing: **archon/x3d/server/parse.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
```

```
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#include <vector>
#include <map>

#include <archon/util/text.H>
#include <archon/util/unicode.H>

#include <archon/x3d/server/script.H>
#include <archon/x3d/server/parse.H>

using namespace std;

namespace Archon
{
  using namespace Utilities;

  namespace X3D
  {
    bool Parser::validateIdentifier(string s)
    {
      ustring t = Unicode::decodeUtf8(s);
      if(!t.size()) return false;
      for(unsigned i=0; i<t.size(); ++i)
      {
        uchar u = t[i];
        if(u >= 0x80) continue;
        if(u <= 0x20 ||
           i == 0 && '0' <= u && u <= '9' ||
           u == '"' || u == '#' || u == '\'' ||
           '+' <= u && u <= '.' ||               //  + , - .
           '[' <= u && u <= ']' ||               //  [ \ ]
           u == '{' || u == '}' || u == 0x7f)
          return false;
      }
      return true;
    }

    void Parser::addNamedNode(string name, Ref<Node> node)
    {
      try
      {
        node->setName(name);
      }
      catch(NodeNameInUseException &)
      {
        warning("Attempt to reuse node name '" + name + "'");
      }
    }

    void Parser::addRootGroupChild(Ref<ChildNode> c, Time stamp)
    {
      rootGroup->add(GroupingNode::childrenField, c, stamp);
    }

    void Parser::addRoute(string sourceNodeName, string sourceFieldName,
```



```
                          string targetNodeName, string targetFieldName)
    {
      const FieldBase *sourceField = 0;
      const FieldBase *targetField = 0;

      bool skip = false;

      Ref<Node> sourceNode = lookupNode(sourceNodeName);
      if(!sourceNode)
      {
        warning("Undefined source node name '" + sourceNodeName +
                "' (route ignored)");
        skip = true;
      }
      else
      {
        sourceField = sourceNode->lookupField(sourceFieldName);
        if(!sourceField)
        {
          warning("Source node type '" + sourceNode->getType()->getName() +
                  "' has no field named '" + sourceFieldName +
                  "' (route ignored)");
          skip = true;
        }
        else if(!sourceField->getIsEventSource())
        {
          warning(sourceNode->getType()->getName() + "." +
                  sourceField->getName() +
                  " cannot generate events (route ignored)");
          skip = true;
        }
      }

      Ref<Node> targetNode = lookupNode(targetNodeName);
      if(!targetNode)
      {
        warning("Undefined target node name '" + targetNodeName +
                "' (route ignored)");
        skip = true;
      }
      else
      {
        targetField = targetNode->lookupField(targetFieldName);
        if(!targetField)
        {
          warning("Target node type '" + targetNode->getType()->getName() +
                  "' has no field named '" + targetFieldName +
                  "' (route ignored)");
          skip = true;
        }
        else if(!targetField->getIsEventTarget())
        {
          warning(targetNode->getType()->getName() + "." +
                  targetField->getName() +
                  " cannot receive events (route ignored)");
          skip = true;
        }
      }
```

```
// Self reference
if(sourceField && targetField &&
   sourceNode == targetNode && sourceField == targetField)
{
  const string s = sourceNode->getName() + "." + sourceField->getName
       ();
  warning("Illegal route, source field '" + s +
          "' and target field '" + s + "' is the same (route ignored)"
               );
  skip = true;
}

const FieldType *fieldType = 0;

// Type conformance
if(sourceField && targetField)
{
  fieldType = sourceField->getType();
  const FieldType *targetFieldType   = targetField->getType();

  /**
   * \todo Somehow sombody must check the runtime types when
   * sending nodes (SFNode/MFNode) across routes.
   */
  if(fieldType != targetFieldType)
  {
    warning("Type mismatch, source field type is '" +
            fieldType->getName() +
            "' and target field type is '" +
            targetFieldType->getName() + "' (route ignored)");
    skip = true;
  }

  /**
   * \todo The below restriction is not mentioned in the X3D
   * specification.
   */
  if(const NodeFieldBase *sourceNodeField =
     dynamic_cast<const NodeFieldBase *>(sourceField))
  {
    const NodeFieldBase *targetNodeField =
      dynamic_cast<const NodeFieldBase *>(targetField);
    if(!sourceNodeField->getNodeType()->
       isDerivedFrom(targetNodeField->getNodeType()))
    {
      warning("Type mismatch, source field type is '" +
              sourceNodeField->getNodeType()->getName() +
              "' and target field type is '" +
              targetNodeField->getNodeType()->getName() +
              "' (route ignored)");
      skip = true;
    }
  }
}

if(skip) return;
```



```
      Route::add(sourceNode, sourceField, targetNode, targetField);

      /*
      logger->log("ROUTE:" + fieldType->getName() +
                  " " + sourceNode->getName() +
                  ":" + sourceNode->getType()->getName() +
                  "." + sourceField->getName() +
                  " TO " + targetNode->getName() +
                  ":" + targetNode->getType()->getName() +
                  "." + targetField->getName());
      */
    }


    int Parser::getNextCustomFieldIndex(Ref<CustomFieldNode> n)
    {
      return n->getNumberOfCustomFields();
    }

    void Parser::addScriptField(Ref<CustomFieldNode> n, Ref<CustomFieldBase>
        f)
    {
      if(!n->addCustomField(f))
        warning("Name of new field '" + f->getName() + "' clashes with "
                "existing fields of the " + n->getType()->getName() +
                " node (field ignored)");
    }
  }
}
```

Listing: **archon/x3d/server/parse_xml.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_PARSE_XML_H
#define ARCHON_X3D_PARSE_XML_H

#include <string>

#include <archon/util/logger.H>
#include <archon/util/ref.H>
```

```
#include <archon/util/stream.H>
#include <archon/util/uri.H>

#include <archon/x3d/server/core.H>
#include <archon/x3d/server/parse.H>

namespace Archon
{
  namespace X3D
  {
    namespace XML
    {
      using namespace std;
      using namespace Utilities;

      struct CDATANode
      {
        /**
         * Must be reentrant.
         */
        virtual void handleCDATA(string, Time stamp) = 0;
      };

      /**
       * Parse a sequence of X3D ChildNodes in XML syntax and add them
       * to the specified GroupingNode. The nodes are instantiated in
       * the specified context.
       *
       * \param progressive If true then child nodes are added to
       * parent nodes as sonn as possible, allowing a progressive load
       * while viewing.
       */
      void parse(Ref<ExecutionContext>, Ref<GroupingNode>, Ref<Stream::Reader
           >,
                 bool progressive=false, Logger * =0, string sourceName="",
                 ProgressTracker * =0);
    }
  }
}

#endif // ARCHON_X3D_PARSE_XML_H
```

Listing: **archon/x3d/server/parse_xml.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
```



```
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

/**
 * \file parse_xml.H
 *
 * Unicode: Note that 'expat' is a unicode compliant XML parser. It
 * supports all of the following character encodings:
 *
 * utf-8
 * utf-16
 * iso-8859-1
 * us-ascii
 *
 * Regardless of the encoding of the input it delivers utf-8 encoded
 * text through its 'callback' functions.
 *
 * \sa http://www.jclark.com/xml/expat.html
 *
 * \todo Verify that the rest of the "glue" code of this file supports
 * unicode.
 */
#include <math.h>

#include <exception>
#include <string>
#include <vector>
#include <map>

// XML parser library
#include <expat.h>

#include <archon/util/term.H>
#include <archon/util/time.H>
#include <archon/util/parse_values.H>
#include <archon/util/text.H>
#include <archon/util/image.H>

#include <archon/x3d/server/field_type.H>
#include <archon/x3d/server/custom_field.H>
#include <archon/x3d/server/server.H>
#include <archon/x3d/server/parse_xml.H>

using namespace std;

namespace Archon
{
  using namespace Math;
  using namespace Utilities;

  namespace X3D
  {
    namespace XML
    {
      struct Parser: X3D::Parser
      {
```

```
XML_Parser xmlParser;
ProgressTracker *progressTracker;
Time creationTimeStamp;
bool progressive;

enum Element
{
  elem_nil,      // An unrecognized or invalid element
  elem_toplevel, // The special element always in bottom of the stack
  elem_x3d,
  elem_head,
  elem_meta,
  elem_component,
  elem_scene,
  elem_route,
  elem_proto,
  elem_field,
  elem_node,     // An element corresponding to a X3D node
  elem_use       // An element exploiting the USE attribute
};

struct StackEntry
{
  Element element;

  /**
   * Is nil if unrecogized or special element (incl. node
   * elements with the USE attribute)
   */
  Ref<Node> node;

  /**
   * Scripts and posibly also prototypes need this instead of
   * a node.
   */
  Ref<CustomFieldBase> field;

  const FieldBase *containerField;

  int containerStackIndex;

  /**
   * Text occuring between start and end tags for this element
   * (not including cdata from sub elements)
   */
  string cdata;

  StackEntry(): element(elem_nil), containerField(0),
       containerStackIndex(-1) {}
};

string getElementName(Element e, const NodeType *t = 0)
{
  switch(e)
  {
  case elem_nil:       return "(nil)";
  case elem_toplevel:  return "(toplevel)";
  case elem_x3d:       return "X3D";
```



```
  case elem_head:      return "head";
  case elem_meta:      return "meta";
  case elem_component: return "component";
  case elem_scene:     return "Scene";
  case elem_route:     return "ROUTE";
  case elem_proto:     return "PROTO";
  case elem_field:     return "field";
  case elem_node:      return t ? t->getName() : "(node)";
  case elem_use:       return t ? t->getName() : "(use)";
  }
  ARCHON_THROW1(ArgumentException,
               "Invalid element type");
}

string getElementName(const StackEntry &e)
{
  return getElementName(e.element, e.node ? e.node->getType() : 0);
}

/**
 * \todo Should be changed so that initially a case sensitive
 * search is done. Then afterwards, if not found, a case
 * insensitive search should be done for maximum tollerance
 * but without loosing specification conformanse.
 *
 * Returns elem_nil for an unknown or unimplemented element
 */
Element elementLookup(string name)
{
  if(Text::compareIgnoreCase(name, "X3D"      )==0) return elem_x3d;
  if(Text::compareIgnoreCase(name, "head"     )==0) return elem_head;
  if(Text::compareIgnoreCase(name, "meta"     )==0) return elem_meta;
  if(Text::compareIgnoreCase(name, "component")==0) return
       elem_component;
  if(Text::compareIgnoreCase(name, "Scene"    )==0) return elem_scene
       ;
  if(Text::compareIgnoreCase(name, "ROUTE"    )==0) return elem_route
       ;
  if(Text::compareIgnoreCase(name, "PROTO"    )==0) return elem_proto
       ;
  if(Text::compareIgnoreCase(name, "field"    )==0) return elem_field
       ;
  return elem_nil;
}

vector<StackEntry> containmentStack;

void warning(string message)
{
  if(logger)
    logger->log(sourceName + ":" + Text::toString(
         XML_GetCurrentLineNumber(xmlParser)) + ": [Warning] "
               + message);
}

Parser(Ref<ExecutionContext> context, Ref<GroupingNode> rootGroup,
       string sourceName, XML_Parser xmlParser,
       ProgressTracker *progressTracker, Logger *logger,
```

```
      Time creationTimeStamp, bool progressive):
  X3D::Parser(context, rootGroup, sourceName, logger),
  xmlParser(xmlParser), progressTracker(progressTracker),
  creationTimeStamp(creationTimeStamp), progressive(progressive)
{
  StackEntry stackEntry;
  stackEntry.element = elem_toplevel;
  containmentStack.push_back(stackEntry);
  initialize();
}


/**
 * Does the string consist of whitespace characters only?
 */
bool isWhiteSpace(const char *s)
{
  while(*s == ' ' || *s == '\t' || *s == '\n' ||
        *s == '\r' || *s == '\f' || *s == ',') ++s;
  return !*s;
}

/**
 * Skip whitespace, then return false if there is no more
 * characters. Otherwise return true and set 'from' to the start of
 * the word and 's' to the character imediately following the
 * last non-whitespace character.
 *
 * Note: We include , as a whitespace character. This seems to
 * be in line with the X3D specification.
 */
bool eatWord(const char *&s, const char *&from)
{
  const char *_s = s;
  while(*_s == ' ' || *_s == '\t' || *_s == '\n' ||
        *_s == '\r' || *_s == '\f' || *_s == ',') ++_s;
  if(!*_s) return false;
  from = _s;
  ++_s;
  while(*_s && *_s != ' ' && *_s != '\t' && *_s != '\n' &&
        *_s != '\r' && *_s != '\f' && *_s != ',') ++_s;
  s = _s;
  return true;
}

bool eat(const char *&s, bool &v, bool &more)
{
  try
  {
    const char *from;
    if(!eatWord(s, from)) return (more = false);
    v = ParseValues::parseBoolean(false, from, s);
  }
  catch(ParseValues::Exception e)
  {
    warning(e.getMessage() + "(assuming false)");
    v = false;
  }
```



```
  return true;
}

bool eat(const char *&s, int &v, bool &more)
{
  try
  {
    const char *from;
    if(!eatWord(s, from)) return (more = false);
    v = ParseValues::parseInteger(true, from, s);
  }
  catch(ParseValues::Exception e)
  {
    warning(e.getMessage() + "(assuming zero)");
    v = 0;
  }
  return true;
}

bool eat(const char *&s, double &v, bool &more)
{
  try
  {
    const char *from;
    if(!eatWord(s, from)) return (more = false);
    v = ParseValues::parseDouble(from, s);
  }
  catch(ParseValues::Exception e)
  {
    warning(e.getMessage() + "(assuming zero)");
    v = 0;
  }
  return true;
}

bool eat(const char *&s, Vector2 &v, bool &more)
{
  try
  {
    const char *from;
    if(!eatWord(s, from)) return (more = false);
    v[0] = ParseValues::parseDouble(from, s);
    if(!eatWord(s, from))
      ARCHON_THROW1(ParseValues::ArgumentException, "2nd component
           missing");
    v[1] = ParseValues::parseDouble(from, s);
  }
  catch(ParseValues::Exception e)
  {
    warning(e.getMessage() + "(assuming zero)");
    v = Vector2::zero();
  }
  return true;
}

bool eat(const char *&s, Vector3 &v, bool &more)
{
  try
```

```
    {
      const char *from;
      if(!eatWord(s, from)) return (more = false);
      v[0] = ParseValues::parseDouble(from, s);
      if(!eatWord(s, from))
        ARCHON_THROW1(ParseValues::ArgumentException, "2nd component
             missing");
      v[1] = ParseValues::parseDouble(from, s);
      if(!eatWord(s, from))
        ARCHON_THROW1(ParseValues::ArgumentException, "3rd component
             missing");
      v[2] = ParseValues::parseDouble(from, s);
    }
    catch(ParseValues::Exception e)
    {
      warning(e.getMessage() + "(assuming zero)");
      v = Vector3::zero();
    }
    return true;
  }

  bool eat(const char *&s, Vector4 &v, bool &more)
  {
    try
    {
      const char *from;
      if(!eatWord(s, from)) return (more = false);
      v[0] = ParseValues::parseDouble(from, s);
      if(!eatWord(s, from))
        ARCHON_THROW1(ParseValues::ArgumentException, "2nd component
             missing");
      v[1] = ParseValues::parseDouble(from, s);
      if(!eatWord(s, from))
        ARCHON_THROW1(ParseValues::ArgumentException, "3rd component
             missing");
      v[2] = ParseValues::parseDouble(from, s);
      if(!eatWord(s, from))
        ARCHON_THROW1(ParseValues::ArgumentException, "4th component
             missing");
      v[3] = ParseValues::parseDouble(from, s);
    }
    catch(ParseValues::Exception e)
    {
      warning(e.getMessage() + "(assuming zero)");
      v = Vector4::zero();
    }
    return true;
  }

  bool eat(const char *&s, AxisRot3 &v, bool &more)
  {
    try
    {
      const char *from;
      if(!eatWord(s, from)) return (more = false);
      v.normalizedAxis[0] = ParseValues::parseDouble(from, s);
      if(!eatWord(s, from))
```



```
        ARCHON_THROW1(ParseValues::ArgumentException, "y component
             missing");
      v.normalizedAxis[1] = ParseValues::parseDouble(from, s);
      if(!eatWord(s, from))
        ARCHON_THROW1(ParseValues::ArgumentException, "z component
             missing");
      v.normalizedAxis[2] = ParseValues::parseDouble(from, s);
      if(!eatWord(s, from))
        ARCHON_THROW1(ParseValues::ArgumentException, "angle missing");
      v.angle = ParseValues::parseDouble(from, s);
    }
    catch(ParseValues::Exception e)
    {
      warning(e.getMessage() + "(assuming no rotation)");
      v = AxisRot3::zero();
    }
    return true;
  }


  bool eat(const char *&s, Image &v, bool &more)
  {
    ARCHON_THROW1(InternalException,
                  "Parsing of inlined images is not supported yet");
  }

  bool eat(const char *&s, Time &v, bool &more)
  {
    double w;
    if(!eat(s, w, more)) return false;
    v = Time(w);
    return true;
  }


  template<class T>
  bool parseValue(const char *s, T &v)
  {
    T w;
    bool m;
    if(!eat(s, w, m))
    {
      warning("Empty string");
      return false;
    }
    if(!isWhiteSpace(s)) warning("Trailing garbage");
    v = w;
    return true;
  }

  template<class T>
  bool parseValue(const char *s, vector<T> &l)
  {
    l.clear();
    T v;
    bool m = true;
    while(m) if(eat(s, v, m)) l.push_back(v);
    return true;
```

```
}

/**
 * We need special handling of SFString fields because
 * anything is valid.
 */
bool parseValue(const char *s, string &v)
{
  v = s;
  return true;
}

/**
 * \todo How is MFString represented in the XML encoding?
 */
bool parseValue(const char *s, vector<string> &v)
{
  for(;;)
  {
    while(*s == ' ' || *s == '\t' || *s == '\n' ||
          *s == '\r' || *s == '\f' || *s == ',') ++s;
    if(!*s) break;
    if(*s == '"' || *s == '\'')
    {
      const char *to = strchr(s+1, *s);
      ++s;
      int n = to ? to - s : strlen(s);
      v.push_back(string(s, n));
      if(!to)
      {
        warning("Unpaired quotation");
        break;
      }
      s += n+1;
      continue;
    }

    const char *to = s+1;
    while(*to && *to != '"' && *to != '\'') ++to;
    --to;
    while(*to == ' ' || *to == '\t' || *to == '\n' ||
         *to == '\r' || *to == '\f' || *to == ',') --to;
    ++to;
    int n = to - s;
    v.push_back(string(s, n));
    s = to;
  }

  return true;
}



struct ValueParserBase
{
  virtual void parse(const char *, Node *,
                     const FieldBase *, Parser *) const = 0;
```



```
  virtual Ref<CustomFieldBase>
  newCustomField(CustomFieldNode *c,
                 int i, string n, const FieldType *t,
                 bool isEventTarget, bool isEventSource,
                 Parser *p, const char *s) const = 0;

protected:
  virtual ~ValueParserBase() {}
};

template<typename T>
struct SingleValueParser: ValueParserBase
{
  void parse(const char *s, Node *n, const FieldBase *f,
             Parser *p) const
  {
    T v;
    if(p->parseValue(s, v))
    {
      Ref<SimpleValue<T> > w = new SimpleValue<T>(f->getType(), v);
      Event e(w.get(), p->creationTimeStamp);

      /*
       * Note that we do not want the value assignment to
       * initiate an event cascade, but this is guaranteed
       * since this field is brand new and cannot have any
       * routes added to it yet. This is because we are
       * currently parsing the attributes of the start tag of
       * the node in question.
       */
      n->handleEvent(f, &e);
    }
  }

  Ref<CustomFieldBase>
  newCustomField(CustomFieldNode *c,
                 int i, string n, const FieldType *t,
                 bool isEventTarget, bool isEventSource,
                 Parser *p, const char *s) const
  {
    T v;
    if(s && p->parseValue(s, v))
    {
      return new SimpleCustomField<T>(c, i, n, t, isEventTarget,
                                      isEventSource, v);
    }
    return new SimpleCustomField<T>(c, i, n, t, isEventTarget,
                                    isEventSource, defaultValue);
  }

  SingleValueParser(const T &v): defaultValue(v) {}

private:
  T defaultValue;
};

template<typename T>
struct SequenceValueParser: ValueParserBase
```

```
{
  typedef vector<T> V;

  void parse(const char *s, Node *n, const FieldBase *f,
             Parser *p) const
  {
    V v;
    if(p->parseValue(s, v))
    {
      Ref<SimpleValue<V> > w = new SimpleValue<V>(f->getType(), v);
      Event e(w.get(), p->creationTimeStamp);

      /*
       * Note that we do not want the value assignment to
       * initiate an event cascade, but this is guaranteed
       * since this field is brand new and cannot have any
       * routes added to it yet. This is because we are
       * currently parsing the attributes of the start tag of
       * the node in question.
       */
      n->handleEvent(f, &e);
    }
  }

  Ref<CustomFieldBase>
  newCustomField(CustomFieldNode *c,
                 int i, string n, const FieldType *t,
                 bool isEventTarget, bool isEventSource,
                 Parser *p, const char *s) const
  {
    V v;
    if(s && p->parseValue(s, v))
    {
      return new SimpleSeqCustomField<T>(c, i, n, t, isEventTarget,
                                         isEventSource, v);
    }
    return new SimpleSeqCustomField<T>(c, i, n, t, isEventTarget,
                                       isEventSource, defaultValue);
  }

private:
  V defaultValue;
};

map<const FieldType *, const ValueParserBase *> valueParsers;

map<string, string> compatibilityFieldTypeNameMap;

void initialize()
{
  Vector4 one4;
  one4[0] = one4[1] = one4[2] = one4[3] = 1;

  valueParsers[SFBool::type]      = new SingleValueParser<bool>(false
      );
  valueParsers[MFBool::type]      = new SequenceValueParser<bool>;
```



```
  valueParsers[SFColor::type]     = new SingleValueParser<Vector3>(
      Vector3(1, 1, 1));
  valueParsers[MFColor::type]     = new SequenceValueParser<Vector3>;

  valueParsers[SFColorRGBA::type] = new SingleValueParser<Vector4>(
      one4);
  valueParsers[MFColorRGBA::type] = new SequenceValueParser<Vector4>;

  valueParsers[SFDouble::type]    = new SingleValueParser<double>(0);
  valueParsers[MFDouble::type]    = new SequenceValueParser<double>;

  valueParsers[SFFloat::type]     = new SingleValueParser<double>(0);
  valueParsers[MFFloat::type]     = new SequenceValueParser<double>;

  valueParsers[SFImage::type]     = new SingleValueParser<Image>(
      Image());
  valueParsers[MFImage::type]     = new SequenceValueParser<Image>;

  valueParsers[SFInt32::type]     = new SingleValueParser<int>(0);
  valueParsers[MFInt32::type]     = new SequenceValueParser<int>;

  valueParsers[SFRotation::type]  = new SingleValueParser<AxisRot3>(
      AxisRot3(Vector3(0, 0, 1), 0));
  valueParsers[MFRotation::type]  = new SequenceValueParser<AxisRot3
      >;

  valueParsers[SFString::type]    = new SingleValueParser<string>(
      string());
  valueParsers[MFString::type]    = new SequenceValueParser<string>;

  valueParsers[SFTime::type]      = new SingleValueParser<Time>(Time
      ());
  valueParsers[MFTime::type]      = new SequenceValueParser<Time>;

  valueParsers[SFVec2d::type]     = new SingleValueParser<Vector2>(
      Vector2(0, 0));
  valueParsers[MFVec2d::type]     = new SequenceValueParser<Vector2>;

  valueParsers[SFVec2f::type]     = new SingleValueParser<Vector2>(
      Vector2(0, 0));
  valueParsers[MFVec2f::type]     = new SequenceValueParser<Vector2>;

  valueParsers[SFVec3d::type]     = new SingleValueParser<Vector3>(
      Vector3(0, 0, 0));
  valueParsers[MFVec3d::type]     = new SequenceValueParser<Vector3>;

  valueParsers[SFVec3f::type]     = new SingleValueParser<Vector3>(
      Vector3(0, 0, 0));
  valueParsers[MFVec3f::type]     = new SequenceValueParser<Vector3>;


  compatibilityFieldTypeNameMap["Boolean"]           = "SFBool";
  compatibilityFieldTypeNameMap["Booleans"]          = "MFBool";
  compatibilityFieldTypeNameMap["Color"]             = "SFColor";
  compatibilityFieldTypeNameMap["Colors"]            = "MFColor";
  compatibilityFieldTypeNameMap["Double"]            = "SFDouble";
  compatibilityFieldTypeNameMap["Doubles"]           = "MFDouble";
  compatibilityFieldTypeNameMap["Float"]             = "SFFloat";
```

```
  compatibilityFieldTypeNameMap["Floats"]             = "MFFloat";
  compatibilityFieldTypeNameMap["Image"]              = "SFImage";
  compatibilityFieldTypeNameMap["Images"]             = "MFImage";
  compatibilityFieldTypeNameMap["Integer"]            = "SFInt32";
  compatibilityFieldTypeNameMap["Integers"]           = "MFInt32";
  compatibilityFieldTypeNameMap["Node"]               = "SFNode";
  compatibilityFieldTypeNameMap["Nodes"]              = "MFNode";
  compatibilityFieldTypeNameMap["Rotation"]           = "SFRotation";
  compatibilityFieldTypeNameMap["Rotations"]          = "MFRotation";
  compatibilityFieldTypeNameMap["String"]             = "SFString";
  compatibilityFieldTypeNameMap["Strings"]            = "MFString";
  compatibilityFieldTypeNameMap["Time"]               = "SFTime";
  compatibilityFieldTypeNameMap["Times"]              = "MFTime";
  compatibilityFieldTypeNameMap["Vector2Float"]       = "SFVec2f";
  compatibilityFieldTypeNameMap["Vector2FloatArray"]  = "MFVec2f";
  compatibilityFieldTypeNameMap["Vector3Float"]       = "SFVec3f";
  compatibilityFieldTypeNameMap["Vector3FloatArray"]  = "MFVec3f";
  compatibilityFieldTypeNameMap["Vector2Double"]      = "SFVec2f";
  compatibilityFieldTypeNameMap["Vector2DoubleArray"] = "MFVec2f";
  compatibilityFieldTypeNameMap["Vector3Double"]      = "SFVec3d";
  compatibilityFieldTypeNameMap["Vector3DoubleArray"] = "MFVec3d";
}

Ref<Node> lookupUseNode(string name, const NodeType *type)
{
  Ref<Node> node = lookupNode(name);
  if(!node)
  {
    warning("Undefined name '" + name + "' in 'USE'");
    return 0;
  }
  if(node->getType() != type)
  {
    warning("Attempt to use '" + node->getType()->getName() + "' as
          '" + type->getName() + "'");
    return 0;
  }
  return node;
}

Ref<Node> checkUseAttribute(const char *atts[], const NodeType *
     nodeType)
{
  for(int i=0; atts[i] && atts[i+1]; i+=2)
  {
    if(Text::compareIgnoreCase(atts[i], "USE") == 0)
    {
      if(string(atts[i]) != "USE")
        warning("Attribute '" + string(atts[i]) + "' should be
             spelled 'USE'");

      if(i>0 || atts[i+2] && atts[i+3])
        warning("Other attributes are not accepted in conjunction
             with 'USE'");
      return lookupUseNode(atts[i+1], nodeType);
    }
  }
  return 0;
```



```
}

void displayStack(bool push)
{
  vector<double> columnWidthFractions;
  columnWidthFractions.push_back(2);
  columnWidthFractions.push_back(5);
  columnWidthFractions.push_back(10);
  columnWidthFractions.push_back(10);
  columnWidthFractions.push_back(2);
  columnWidthFractions.push_back(2);
  columnWidthFractions.push_back(10);
  columnWidthFractions.push_back(10);
  Text::Table<string> table(containmentStack.size()+1,
       columnWidthFractions);
  for(unsigned i=0; i<containmentStack.size(); ++i)
  {
    StackEntry &e = containmentStack[i];
    table(i, 0) = Text::toString(i);
    table(i, 1) = getElementName(e.element);
    table(i, 2) = e.node ? e.node->getType()->getName() : "";
    table(i, 3) = e.node ? e.node->getName() : "";
    table(i, 4) = "->";
    table(i, 5) = Text::toString(e.containerStackIndex);
    table(i, 6) = e.containerField ? e.containerField->getName() : ""
         ;
    if(const NodeFieldBase *f =
       dynamic_cast<const NodeFieldBase *>(e.containerField))
      table(i, 7) = f->getNodeType()->getName();
    else
      table(i, 7) = "";
  }

  logger->log("Stack " + string(push?"push":"pop") + ":\n" +
              Text::format(table, Term::getWidth(), 2, false));
}

bool attr_cmp(string s, string attribute, Element element,
              bool deprecated = false)
{
  if(Text::compareIgnoreCase(s, attribute)==0)
  {
    if(s != attribute)
      warning("Attribute '" + s + "' of element '" +
              getElementName(element) + "' should be spelled '" +
              attribute + "'");
    if(deprecated)
      warning("Use of attribute '" + s + "' in element '" +
              getElementName(element) + "' is deprecated.");
    return true;
  }
  return false;
}

/*
 * Rules for verification of element containment:
 *
 * The X3D element may be contained in the imaginary top-level
```

```
 * element only.
 *
 * The special non-node hierachical elements such as head, meta,
 * component & Scene may never be contained in any node-type
 * element.
 *
 * Route and Proto elements may be contained in the scene element
 * or in any element contained directly or indirectly in the scene
 * element.
 *
 * Node-type elements may be contained in other node-type elements
 * according to their runtime type information only with the
 * exception that those derived from the ChildNode may be
 * contained in the special Scene element.
 *
 * Nothing may be contained in a node-type element that utilizes
 * the USE attribute.
 *
 * When verifying the containment rules, the containment stack is
 * consulted. The element at the top of the stack is always the
 * element that imediately encloses the currently encountered
 * element. If this element at the top of the stack is a valid
 * container for the new element, then all is fine. If not, then
 * the stack is traversed from top to bottom, to see if any
 * non-immediately enclosing element complies as a container. If
 * such a container is found then it will be used but a warning
 * will be issued.
 *
 * Any encountered element that does not have a compliant
 * container on the stack will be ignored
 */
void parseStartElement(const char *name, const char *atts[],
                       StackEntry *stackEntry)
{
  // Determine the type of element
  Element element = elementLookup(name);
  const NodeType *nodeType = 0;

  if(element == elem_nil)
  {
    nodeType = NodeType::lookup(name);

    if(!nodeType || !nodeType->isUserInstantiable())
    {
      warning("Unknown element '" + string(name) + "' (element
           ignored)");
      return;
    }

    if(nodeType->getName() != name)
      warning("Element '" + string(name) + "' should be spelled '" +
              nodeType->getName() + "'");

    element = elem_node; // May later become 'elem_use'
  }
  else if(getElementName(element) != name)
    warning("Element '" + string(name) + "' should be spelled '" +
            getElementName(element) + "'");
```



```
  /*
   * Find the container element (and the target field for
   * node-type elements):
   *
   * See the note on containment verification rules above!
   *
   * When the search terminates 'containerStackIndex == -1'
   * indicates 'not found', else if 'containerField == 0' the the
   * new element is a child of the Scene element.
   */
  int containerStackIndex = -1;
  const FieldBase *containerField = 0;
  bool fallThrough = false;
  for(int i = int(containmentStack.size())-1; i>=0; --i)
  {
    if(i < int(containmentStack.size())-1) fallThrough = true;

    Element containerElement = containmentStack[i].element;

    // Get rid of impossible containers
    if(containerElement == elem_nil ||
       containerElement == elem_route ||
       containerElement == elem_proto ||
       containerElement == elem_use) continue;

    // Handle new node-type elements
    if(element == elem_node)
    {
      if(containerElement == elem_field)
      {
        const FieldType *t = containmentStack[i].field->getType();
        if(t == SFNode::type || t == MFNode::type)
        {
          containerStackIndex = containmentStack[i].
               containerStackIndex;
          containerField = containmentStack[i].field.get();
          break;
        }
        continue;
      }

      if(containerElement == elem_scene &&
         nodeType->isDerivedFrom(ChildNode::type))
      {
        containerStackIndex = i;
        break;
      }

      if(containerElement != elem_node) break;

      Ref<Node> n = containmentStack[i].node;
      if(!n) ARCHON_THROW1(InternalException,
                           "x3d/parse_xml: Stack entry "
                           "unexpectedly had no node");
      const NodeType *t = n->getType();
```

```
    containerField = t->lookupNodeField(nodeType);
    if(containerField)
    {
      containerStackIndex = i;
      break;
    }
    continue;
  }

  // Handle Script fields
  if(element == elem_field)
  {
    if(containerElement != elem_node) continue;

    Ref<Node> n = containmentStack[i].node;
    if(!n) ARCHON_THROW1(InternalException,
                         "XML::Parser::parseStartElement: "
                         "Stack entry unexpectedly had no "
                         "node");
    if(!dynamic_cast<CustomFieldNode *>(n.get())) continue;
    containerStackIndex = i;
    break;
  }

  // Handle Route and PROTO
  if(element == elem_route || element == elem_proto)
  {
    if(containerElement == elem_scene ||
       containerElement == elem_node)
    {
      containerStackIndex = i;
      break;
    }
    continue;
  }

  // Handle special non-node hierachical elements
  if(element == elem_x3d && containerElement == elem_toplevel ||
     element == elem_head && containerElement == elem_x3d ||
     element == elem_meta && containerElement == elem_head ||
     element == elem_component && containerElement == elem_head ||
     element == elem_scene && containerElement == elem_x3d)
  {
    containerStackIndex = i;
    break;
  }
}


/*
 * If we didn't find a compatible container element or
 * the container element wasn't the imediately enclosing element
 *
 * Note: this should be converted to a switch staement!
 */
if(containerStackIndex < 0 || fallThrough)
{
  string message;
```



```
  if(containmentStack.back().element == elem_toplevel)
    message = "Element '" + getElementName(element, nodeType) +
      "' is not allowed as the top level element";
  else if(containmentStack.back().element == elem_use)
    message = "Element '" + getElementName(element, nodeType) +
      "' is not allowed in elements that employ the USE "
      "attribute";
  else if(containmentStack.back().element == elem_nil)
    message = "Ill placed element '" +
      getElementName(element, nodeType) + "'";
  else if(containmentStack.back().element == elem_field)
    message = "Node elements are not allowed in field elements "
      "of type '" +
      containmentStack.back().field->getType()->getName() + "'";
  else
    message = "Element '" + getElementName(element, nodeType) +
      "' is not allowed in '" +
      getElementName(containmentStack.back()) + "'";

  if(containerStackIndex == -1)
  {
    warning(message + " (element ignored)");
    return;
  }

  warning(message + " (fall-through to closest '" +
          getElementName(containmentStack[containerStackIndex]) +
          "' established)");
}


Ref<Node> node;
Ref<CustomFieldBase> field;

if(element == elem_route)
{
  string fromNode;
  string fromField;
  string toNode;
  string toField;
  bool gotFromNode  = false;
  bool gotFromField = false;
  bool gotToNode    = false;
  bool gotToField   = false;

  for(int i=0; atts[i] && atts[i+1]; i+=2)
  {
    if(attr_cmp(atts[i], "fromNode", element))
    {
      gotFromNode = parseValue(atts[i+1], fromNode);
    }
    else if(attr_cmp(atts[i], "fromField", element))
    {
      gotFromField = parseValue(atts[i+1], fromField);
    }
    else if(attr_cmp(atts[i], "toNode", element))
    {
      gotToNode = parseValue(atts[i+1], toNode);
```

```
    }
    else if(attr_cmp(atts[i], "toField", element))
    {
      gotToField = parseValue(atts[i+1], toField);
    }
    else
    {
      warning("Unrecognized attribute '" + string(atts[i]) +
              "' of element '" + getElementName(element) + "' (
                  attribute ignored)");
    }
  }

  bool skip = false;
  if(!gotFromNode)
  {
    warning("Required attribute 'fromNode' of element '" +
            getElementName(element) + "' is missing (element
                ignored)");
    skip = true;
  }
  if(!gotFromField)
  {
    warning("Required attribute 'fromField' of element '" +
            getElementName(element) + "' is missing (element
                ignored)");
    skip = true;
  }
  if(!gotToNode)
  {
    warning("Required attribute 'toNode' of element '" +
            getElementName(element) + "' is missing (element
                ignored)");
    skip = true;
  }
  if(!gotToField)
  {
    warning("Required attribute 'toField' of element '" +
            getElementName(element) + "' is missing (element
                ignored)");
    skip = true;
  }
  if(!skip) addRoute(fromNode, fromField, toNode, toField);
}
else if(element == elem_proto)
{
  warning("Element '" + getElementName(element) + "' is not
      supported yet (ignored)");
}
else if(element == elem_field)
{
  string name;
  string accessType;
  string type;
  string use;

  bool gotName       = false;
  bool gotAccessType = false;
```



```
  bool gotType       = false;
  int  gotValue      = -1; // If >= 0 value is atts[gotValue]
  bool gotUse        = false;

  for(int i=0; atts[i] && atts[i+1]; i+=2)
  {
    if(attr_cmp(atts[i], "name", element))
    {
      gotName = parseValue(atts[i+1], name);
    }
    else if(attr_cmp(atts[i], "accessType", element))
    {
      gotAccessType = parseValue(atts[i+1], accessType);
    }
    else if(attr_cmp(atts[i], "type", element))
    {
      gotType = parseValue(atts[i+1], type);
    }
    else if(attr_cmp(atts[i], "value", element))
    {
      gotValue = i+1;
    }
    else if(attr_cmp(atts[i], "USE", element, true))
    {
      // This attribute is deprecated
      gotUse = parseValue(atts[i+1], use);
    }
    else
    {
      warning("Unrecognized attribute '" + string(atts[i]) +
              "' of element '" + getElementName(element) + "' (
                  attribute ignored)");
    }
  }

  bool isEventTarget = false;
  bool isEventSource = false;

  bool skip = false;
  if(gotName && !validateIdentifier(name))
  {
    warning("Illegal value '" + name + "' of attribute 'name' "
            "in element 'field' (element ignored)");
    skip = true;
  }

  if(gotAccessType)
  {
    if(Text::compareIgnoreCase("initializeOnly", accessType
        ) == 0 ||
       Text::compareIgnoreCase("field", accessType) == 0);
    else if(Text::compareIgnoreCase("inputOnly", accessType
        ) == 0 ||
            Text::compareIgnoreCase("eventIn", accessType) == 0)
      isEventTarget = true;
    else if(Text::compareIgnoreCase("outputOnly", accessType
        ) == 0 ||
            Text::compareIgnoreCase("eventOut", accessType) == 0)
```

```
    isEventSource = true;
  else if(Text::compareIgnoreCase("inputOutput", accessType
       ) == 0 ||
          Text::compareIgnoreCase("exposedField", accessType
               ) == 0)
    isEventTarget = isEventSource = true;
  else
  {
    warning("Unrecognized attribute value '" + accessType +
            "' of attribute 'accessType' of element 'field' "
            "(assuming 'initializeOnly')");
    accessType = "initializeOnly";
  }

  if(!isEventTarget && !isEventSource &&
     accessType != "initializeOnly")
    warning("Attribute value '" + accessType + "' of attribute "
            "'accessType' of element 'field' should be spelled "
            "'initializeOnly'");
  if(isEventTarget && !isEventSource && accessType != "inputOnly"
       )
    warning("Attribute value '" + accessType + "' of attribute "
            "'accessType' of element 'field' should be spelled "
            "'inputOnly'");
  if(!isEventTarget && isEventSource && accessType != "outputOnly
       ")
    warning("Attribute value '" + accessType + "' of attribute "
            "'accessType' of element 'field' should be spelled "
            "'outputOnly'");
  if(isEventTarget && isEventSource && accessType != "inputOutput
       ")
    warning("Attribute value '" + accessType + "' of attribute "
            "'accessType' of element 'field' should be spelled "
            "'inputOutput'");
}

const FieldType *fieldType = 0;
if(gotType)
{
  string mappedType = type;
  map<string, string>::iterator i =
    compatibilityFieldTypeNameMap.find(type);
  if(i != compatibilityFieldTypeNameMap.end())
    mappedType = i->second;

  fieldType = FieldType::lookup(mappedType);
  if(!fieldType)
  {
    warning("Unrecognized attribute value '" + type + "' of
         attribute "
            "'type' of element 'field' (element ignored)");
    skip = true;
  }
  else if(fieldType->getName() != type)
    warning("Attribute value '" + type + "' of attribute "
            "'type' of element 'field' should be spelled "
            "'" + fieldType->getName() + "'");
}
```



```
if(!gotName)
{
  warning("Required attribute 'Name' of element '" +
          getElementName(element) + "' is missing (element
               ignored)");
  skip = true;
}
if(!gotAccessType)
  warning("Required attribute 'accessType' of element '" +
          getElementName(element) + "' is missing (assuming '
               initializeOnly')");
if(!gotType)
{
  warning("Required attribute 'type' of element '" +
          getElementName(element) + "' is missing (element
               ignored)");
  skip = true;
}

// Instantiate a field object for the element stack
if(!skip && fieldType)
{
  CustomFieldNode *n = dynamic_cast<CustomFieldNode *>
    (containmentStack[containerStackIndex].node.get());
  if(!n) ARCHON_THROW1(InternalException,
                       "XML::Parser::parseStartElement: "
                       "Custom field container was not a "
                       "CustumFieldNode");
  int customFieldIndex = getNextCustomFieldIndex(n);

  Ref<Node> useNode;
  if(gotUse)
  {
    useNode = lookupNode(use);
    if(!useNode)
      warning("Undefined node name '" + use + "' in 'USE' "
              "attribute of field element");
  }

  if(fieldType == SFNode::type)
    field = new NodeCustomField<Node>
      (n, customFieldIndex, name, Node::type,
       isEventTarget, isEventSource, useNode);
  else if(fieldType == SFNode::type)
  {
    vector<Ref<Node> > v;
    v.push_back(useNode);
    field = new NodeSequenceCustomField<Node>
      (n, customFieldIndex, name, Node::type,
       isEventTarget, isEventSource, v);
  }
  else
  {
    field = valueParsers[fieldType]->newCustomField
      (n, customFieldIndex, name, fieldType,
       isEventTarget, isEventSource,
       this, gotValue >= 0 ? atts[gotValue] : 0);
```

```
        if(useNode)
          warning("The USE attribute may not be used in field "
                  "elements with type '" + fieldType->getName() +
                  "' (attribute ignored)");
      }
    }
  }
  else if(element == elem_node)
  {
    // Is it a "reuse" element refering to an already instantiated
         named node?
    node = checkUseAttribute(atts, nodeType);


    /*
     * Instantiate a node of the desired type if it wasn't a
     * "reuse" element.
     */
    if(node) element = elem_use;
    else node = nodeType->instantiate(context);


    // Parse attributes
    if(element != elem_use) for(int i=0; atts[i] && atts[i+1]; i+=2)
    {
      if(Text::compareIgnoreCase(atts[i], "DEF") == 0)
      {
        if(string(atts[i]) != "DEF")
          warning("Attribute '" + string(atts[i]) + "' should be
               spelled 'DEF'");

        const string name = atts[i+1];
        if(validateIdentifier(name)) addNamedNode(name, node);
        else
          warning("Illegal value '" + name + "' of attribute 'DEF' "
                  "in element '" + nodeType->getName() + "' (
                       attribute ignored)");
      }
      else
      {
        const FieldBase *field = nodeType->lookupField(atts[i]);
        if(!field || field->getType() == SFNode::type || field->
             getType() == MFNode::type)
        {
          warning("Unrecognized attribute '" + string(atts[i]) +
                  "' of element '" + nodeType->getName() + "' (
                       attribute ignored)");
          continue;
        }

        if(field->getName() != atts[i])
          warning("Attribute '" + string(atts[i]) + "' should be
               spelled '" +
                  field->getName() + "'");

        valueParsers[field->getType()]->parse(atts[i+1], node.get(),
             field, this);
      }
```



```
      }
    }

    stackEntry->element = element;
    stackEntry->node = node;
    stackEntry->field = field;
    stackEntry->containerField = containerField;
    stackEntry->containerStackIndex = containerStackIndex;

    if(progressive &&
       (element == elem_node || element == elem_use) &&
       !dynamic_cast<CustomFieldNode *>(node.get()))
      injectChild(stackEntry);
  }

  void startElement(const char *name, const char *atts[])
  {
    StackEntry stackEntry;
    parseStartElement(name, atts, &stackEntry);
    containmentStack.push_back(stackEntry);
    //displayStack(true);
  }

  void endElement()
  {
    StackEntry stackEntry = containmentStack.back();
    containmentStack.pop_back();
    //displayStack(false);

    // Is it a field element?
    if(stackEntry.element == elem_field)
    {
      if(!stackEntry.field) return;
      if(stackEntry.containerStackIndex < 0) return;
      Ref<CustomFieldNode> n = dynamic_cast<CustomFieldNode *>
        (containmentStack[stackEntry.containerStackIndex].node.get());
      if(!n) ARCHON_THROW1(InternalException,
                           "XML::Parser::endElement: "
                           "Custom field container was not a "
                           "CustomFieldNode");

      addScriptField(n, stackEntry.field);

      return;
    }

    // Skip all non-node elements
    if(stackEntry.element != elem_node &&
       stackEntry.element != elem_use) return;

    // Inject XML/CDATA into CDATA nodes
    if(stackEntry.element == elem_node)
      if(CDATANode *n = dynamic_cast<CDATANode *>(stackEntry.node.get
           ()))
      {
        string s = Text::trim(stackEntry.cdata);
        if(s.size()) n->handleCDATA(s, creationTimeStamp);
      }
```

```
    if(!progressive ||
       dynamic_cast<CustomFieldNode *>(stackEntry.node.get()))
      injectChild(&stackEntry);
  }

  void injectChild(StackEntry *stackEntry)
  {
    // Is it a Scene child?
    if(!stackEntry->containerField)
    {
      addRootGroupChild(dynamic_cast<ChildNode *>
                        (stackEntry->node.get()), creationTimeStamp);
      return;
    }

    // Is there room for another node in the target field? Try
    // to inject it!
    bool warn = false;
    if(const NodeFieldBase *f =
       dynamic_cast<const NodeFieldBase *>(stackEntry->containerField))
    {
      if(!containmentStack[stackEntry->containerStackIndex].node->
           inject
         (f, stackEntry->node, creationTimeStamp)) warn = true;
    }

    if(warn)
      warning("Element '" +
              containmentStack[stackEntry->containerStackIndex].node->
              getType()->getName() +
              "' already contains a '" +
              stackEntry->node->getType()->getName() +
              "' (additional element ignored)");
  }

  void characterDataHandler(const char *s, int n)
  {
    StackEntry &stackEntry = containmentStack.back();
    if(dynamic_cast<const CDATANode *>(stackEntry.node.get()))
      stackEntry.cdata.append(s, n);
  }
};

namespace
{
  void startElement(void *userData, const char *name, const char *atts
       []) throw()
  {
    Parser *p = static_cast<Parser *>(userData);
    p->startElement(name, atts);
  }

  void endElement(void *userData, const char *) throw()
  {
    Parser *p = static_cast<Parser *>(userData);
    p->endElement();
  }
```



```
    void characterData(void *userData, const char *s, int n)
    {
      Parser *p = static_cast<Parser *>(userData);
      p->characterDataHandler(s, n);
    }

    struct XmlParserWrapper
    {
      const XML_Parser ptr;

      XmlParserWrapper(XML_Parser ptr):ptr(ptr) {}

      ~XmlParserWrapper()
      {{ // The extra scope is needed to work around gcc3.2 bug #8287
        if(ptr) XML_ParserFree(ptr);
      }}
    };
  }


  void parse(Ref<ExecutionContext> context, Ref<GroupingNode> group,
             Ref<Stream::Reader> reader, bool progressive, Logger *logger
                 ,
             string sourceName, ProgressTracker *progressTracker)
  {
    Ref<Server> server;
    try { server = context->server.getRef(); }
    catch(ForwardDestroyedException &) { return; }

    XmlParserWrapper p(XML_ParserCreate(0));
    XML_SetElementHandler(p.ptr, startElement, endElement);
    XML_SetCharacterDataHandler(p.ptr, characterData);

    Parser parser(context, group,
                  sourceName.empty() ? context->getBaseUri().getFile() :
                  sourceName, p.ptr, progressTracker,
                  logger ? logger : server->getLogger(),
                  server->getNextTimeStamp(), progressive);
    XML_SetUserData(p.ptr, &parser);

    const unsigned bufferSize = 1024;
    char buffer[bufferSize];

    int progress = 0;
    for(;;)
    {
      const int n = reader->read(buffer, bufferSize);
      if(!XML_Parse(p.ptr, buffer, n, n==0))
        ARCHON_THROW1(Parser::ParseException, XML_ErrorString(
             XML_GetErrorCode(p.ptr)));
      if(n==0) break;
      progress += n;
      if(progressTracker) progressTracker->progress(progress);
    }
  }
 }
}
```

```
}
```

Listing: **archon/x3d/server/scene.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_SCENE_H
#define ARCHON_X3D_SCENE_H

/*
 *  X3D ExecutionContext and Scene
 */

#include <vector>
#include <string>

#include <archon/math/vector.H>
#include <archon/math/geometry.H>
#include <archon/util/logger.H>
#include <archon/util/ref.H>

#include <archon/x3d/server/load.H>
#include <archon/x3d/server/time.H>
#include <archon/x3d/server/group.H>
#include <archon/x3d/server/event.H>

namespace Archon
{
  namespace X3D
  {
    using namespace std;
    using namespace Utilities;
    using namespace Math;

    struct Server; // "server.H"

    struct ExecutionContext: virtual BackRefObjectBase
    {
      struct Exception: virtual X3D::Exception {};
      struct ArgumentException: Exception, X3D::ArgumentException
      {
```



```
        ArgumentException(string l, string m): Utilities::Exception(l, m) {}
      };


      /**
       * \param fileServer If not null this refers to a server that
       * will handle file-scheme URIs.
       */
      ExecutionContext(Ref<Server>, Uri baseUri,
                       Ref<Loader::FileServer> fileServer=0);

      ~ExecutionContext();

      /**
       * Should be thrown by methods that are illegal on realized
       * contexts.
       */
      struct RealizedException: X3D::StateException
      {
        RealizedException(string l): Utilities::Exception(l) {}
      };

      /**
       * If this context is not already realized, then realize it now.
       *
       * A context is realized when it is either added as the root
       * context of the server, or when it is added to a SubContextNode.
       */
      void realize();

      bool isRealized() const;

      Ref<Group> getRootGroup() const;
      void setRootGroup(Ref<Group>);

      /**
       * Requires scenegraph read lock.
       */
      Ref<Group> getRootGroupNoLock() { return rootGroup; }

      Uri getBaseUri() const;

      Ref<Node> lookupNode(string name) const;

      /**
       * Requires scenegraph read lock.
       */
      string dump(int level=0) { return rootGroup->dump(level); }

      /**
       * Activate time sensors.
       * Requires scenegraph write lock.
       */
      void tick();

      const BackRef<Server> server;

      /**
```

```
     * If not null this refers to a server that will handle
     * file-scheme URIs.
     */
    const BackRef<Loader::FileServer> fileServer;

  protected:
    /**
     * Main mutex for Contexts and derivatives.
     */
    const Mutex contextMutex;

  private:
    void refForwardDestroy();

    Uri baseUri;
    Ref<Group> rootGroup;
    bool realized;

    friend struct Route;
    /**
     * We need to use a different mutex to protect the 'routes'
     * variable since this method is called from the Route
     * destructors wich are called at times that are hard to
     * predict. Especially it is hard to avoid calling any Route
     * detructor while holding 'mutex'.
     */
    Mutex routesMutex;
    list<Route *> routes;
    void add(Route *);
    void remove(Route *);

    friend struct Node;
    /**
     * We need to use a different mutex to protect the 'namedNodes'
     * variable since this method is called from the Node
     * destructors wich are called at times that are hard to
     * predict. Especially it is hard to avoid calling any Node
     * detructor while holding 'mutex'.
     */
    Mutex namedNodesMutex;
    map<string, Node *> namedNodes;
    void changeNodeName(Node *, string oldName, string newName)
      throw(NodeNameInUseException);

    friend struct TimeSensor;
    /**
     * We need to use a different mutex to protect the 'timeSensors'
     * variable since this method is called from the Node
     * destructors wich are called at times that are hard to
     * predict. Especially it is hard to avoid calling any Node
     * detructor while holding 'mutex'.
     */
    Mutex timeSensorsMutex;
    list<TimeSensor *> timeSensors;
    void addTimeSensor(TimeSensor *);
    void removeTimeSensor(TimeSensor *);
  };
```



```
    struct Scene: ExecutionContext
    {
      Scene(Ref<Server> server, Uri baseUri,
            Ref<Loader::FileServer> fileServer=0);
    };
  }
}

#endif // ARCHON_X3D_SCENE_H
```

Listing: **archon/x3d/server/scene.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

/*
 *  X3D ExecutionContext and Scene
 */

#include <archon/x3d/server/server.H>
#include <archon/x3d/server/scene.H>

namespace Archon
{
  namespace X3D
  {
    ExecutionContext::ExecutionContext(Ref<Server> server, Uri baseUri,
                                       Ref<Loader::FileServer> fileServer):
      server(server), fileServer(fileServer),
      baseUri(baseUri), realized(false)
    {
    }

    ExecutionContext::~ExecutionContext()
    {
      if(!isRealized()) return;
      try { server->remove(this); }
      catch(ForwardDestroyedException &) {}
    }

    void ExecutionContext::refForwardDestroy()
    {
```

```
  rootGroup.reset();
}

void ExecutionContext::realize()
{
  Mutex::Lock l(contextMutex);
  if(realized) return;
  realized = true;
  server->add(this);
}

bool ExecutionContext::isRealized() const
{
  Mutex::Lock l(contextMutex);
  return realized;
}


Ref<Group> ExecutionContext::getRootGroup() const
{
  {
    Mutex::Lock l(contextMutex);
    if(!realized) return rootGroup;
  }
  try
  {
    Mutex::Lock l(server->scenegraphMutex);
    return rootGroup;
  }
  catch(ForwardDestroyedException &) { return 0; }
}

void ExecutionContext::setRootGroup(Ref<Group> g)
{
  if(g)
  {
    if(g->context !=
       BackRef<ExecutionContext>(Ref<ExecutionContext>(this)))
      ARCHON_THROW1(ArgumentException,
                    "setRootGroup: Group has wrong context");
    g->realize();
  }

  {
    Mutex::Lock l(contextMutex);
    if(!realized)
    {
      rootGroup = g;
      return;
    }
  }
  try
  {
    Mutex::Lock l(server->scenegraphMutex);
    rootGroup = g;
  }
  catch(ForwardDestroyedException &) {}
}
```



```
Uri ExecutionContext::getBaseUri() const
{
  Mutex::Lock l(contextMutex);
  return baseUri;
}

Ref<Node> ExecutionContext::lookupNode(string name) const
{
  Mutex::Lock l(namedNodesMutex);
  map<string, Node *>::const_iterator i=namedNodes.find(name);
  if(i==namedNodes.end()) return 0;
  return Ref<Node>(i->second, RefSafeIncTag());
}

void ExecutionContext::add(Route *r)
{
  Mutex::Lock l(routesMutex);
  routes.push_back(r);
}

void ExecutionContext::remove(Route *r)
{
  Mutex::Lock l(routesMutex);
  routes.remove(r);
}

void ExecutionContext::changeNodeName(Node *n, string oldName, string
     newName)
  throw(NodeNameInUseException)
{
  if(newName == oldName) return;
  Mutex::Lock l(namedNodesMutex);
  if(!newName.empty())
  {
    pair<string, Node *> p(newName, n);
    if(!namedNodes.insert(p).second) ARCHON_THROW(NodeNameInUseException
         );
  }
  if(!oldName.empty()) namedNodes.erase(oldName);
}

void ExecutionContext::addTimeSensor(TimeSensor *n)
{
  Mutex::Lock l(timeSensorsMutex);
  timeSensors.push_back(n);
}

void ExecutionContext::removeTimeSensor(TimeSensor *n)
{
  Mutex::Lock l(timeSensorsMutex);
  timeSensors.remove(n);
}

void ExecutionContext::tick()
{
  list<Ref<TimeSensor> > t;
  {
```

```
        Mutex::Lock l(timeSensorsMutex);
        list<TimeSensor *>::iterator i = timeSensors.begin();
        while(i != timeSensors.end())
        {
          Ref<TimeSensor> r(*i, RefSafeIncTag());
          if(r) t.push_back(r);
          ++i;
        }
      }
      {
        list<Ref<TimeSensor> >::iterator i = t.begin();
        while(i != t.end())
        {
          (*i)->tick();
          ++i;
        }
      }
    }

    Scene::Scene(Ref<Server> server, Uri baseUri,
                 Ref<Loader::FileServer> fileServer):
      ExecutionContext(server, baseUri, fileServer)
    {
    }
  }
}
```

Listing: **archon/x3d/server/script.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_SCRIPT_H
#define ARCHON_X3D_SCRIPT_H

/*
 * \file script.H
 *
 *  X3D Scripting component
 */

#include <archon/x3d/server/custom_field.H>
```



```
#include <archon/x3d/server/network.H>
#include <archon/x3d/server/parse_xml.H>

namespace Archon
{
  namespace X3D
  {
    struct Parser;

    struct ScriptNode: virtual ChildNode
    {
      static const NodeType *type;

    protected:
      ScriptNode() {}
    };

    /**
     * Note that the X3D specification does not indicate derivation
     * from UrlObject. But to keep things simple and URL fields
     * consistent across various types of nodes, this is done anyway.
     *
     * Note: The script should start interpreting the source code and
     * preparing the engine no later than when the script node is
     * realized (ie. take from the initialization state to the
     * realized state). It may be impossible to do it earlier since at
     * such a time we won't know if all custom fields have been added
     * or there are more to come.
     */
    struct Script: ScriptNode, UrlObject, CustomFieldNode, XML::CDATANode
    {
      static Ref<Node> instantiate(BackRef<ExecutionContext> c)
      {
        return new Script(c);
      }

      static const NodeType *type;
      const NodeType *getType() const { return type; }

      Script(BackRef<ExecutionContext> c);

      bool getDirectOutput() const { return directOutput; }
      bool getMustEvaluate() const { return mustEvaluate; }


      struct Engine: virtual RefObjectBase
      {
        /**
         * Must be called with a scenegraph write lock, and never with
         * a lock on any node. Also, this method may never be called
         * for an un-realized containing node.
         */
        virtual void handleEvent(int fieldIndex, Time timestamp) = 0;
      };

    private:
      friend void initializeScriptComponent();
```

```
      bool directOutput; // [] FALSE
      bool mustEvaluate; // [] FALSE

      Loader::UriType getUriType() const
      {
        return Loader::uriType_Script;
      }

      void update();
      void onRealized(); // Overriding from Node
      void onLoaded(); // Overriding from UrlObject
      void clear(); // Overriding from UrlObject

      // Overriding from CustomFieldNode
      void handleEvent(int fieldIndex, Time timestamp);

      void handleCDATA(string, Time stamp); // Overriding from CDATANode

      Ref<Engine> engine;
    };
  }
}

#endif // ARCHON_X3D_SCRIPT_H
```

Listing: **archon/x3d/server/script.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

/*
 * \file script.C
 *
 *  X3D Scripting component
 */

#include <archon/x3d/server/field_type.H>
#include <archon/x3d/server/field.H>
#include <archon/x3d/server/server.H>
#include <archon/x3d/server/script.H>
#include <archon/x3d/server/script_ecma.H>
```



```
namespace Archon
{
  namespace X3D
  {
    extern long nodeTypeIndexEnumerator;

    const NodeType *ScriptNode::type = 0;
    const NodeType *Script::type = 0;

    void initializeScriptComponent()
    {
      vector<const FieldBase *> fields;


      // ScriptNode

      ScriptNode::type =
        NodeType::newAbstract("ScriptNode", false, 0,
                              ChildNode::type);


      // Script

      fields.push_back(newPrivateField("directOutput", SFBool::type,
                                       &Script::directOutput));
      fields.push_back(newPrivateField("mustEvaluate", SFBool::type,
                                       &Script::mustEvaluate));
      Script::type =
        NodeType::newConcrete("Script", Script::instantiate, &fields,
                              ScriptNode::type, UrlObject::type);
      fields.clear();
    }


    Script::Script(BackRef<ExecutionContext> c):
      Node(c), directOutput(false), mustEvaluate(false)
    {
    }


    /**
     * \todo Should realize the script node if it is not realized
     * already. This is to prevent any further field additions.
     */
    void Script::update()
    {
      Ref<const Loader::Contents> c = getContents();
      if(!c) return;
      Mutex::Lock l(context->server->scenegraphMutex);
      if(engine) return;
      const Loader::ScriptContents *d =
        dynamic_cast<const Loader::ScriptContents *>(c.get());
      if(!d) ARCHON_THROW1(InternalException,
                           "Script::update: Got non-script "
                           "contents object");
      if(d->language == "ecmascript")
        engine = new ECMA::Engine(this, d->sourceCode, d->sourceName, d->
             sourceLine);
```

```
      else ARCHON_THROW1(InternalException,
                         "Script::update: Got unexpected language");
    }

    void Script::clear()
    {
      engine = 0;
    }

    /**
     * Called asynchroniously, so be very carefull here. See
     * UrlObject::onLoaded.
     */
    void Script::onLoaded()
    {
      if(!isRealized()) return;
      update();
    }

    void Script::onRealized()
    {
      update();
    }

    void Script::handleEvent(int fieldIndex, Time timestamp)
    {
      if(engine) engine->handleEvent(fieldIndex, timestamp);
    }

    void Script::handleCDATA(string s, Time stamp)
    {
      // Default to ECMA Script
      if(!Text::isPrefixOf("ecmascript:", s, true) &&
         !Text::isPrefixOf("javascript:", s, true))
        s = "ecmascript:" + s;
      addUri(s, stamp);
    }
  }
}
```

Listing: **archon/x3d/server/script_ecma.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
```



```
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_SCRIPT_ECMA_H
#define ARCHON_X3D_SCRIPT_ECMA_H

/*
 * \file script.H
 *
 *  X3D Scripting component
 */

#define XP_UNIX
#define DEBUG
#include <jsapi.h>

#include <archon/x3d/server/custom_field.H>
#include <archon/x3d/server/script.H>

namespace Archon
{
  namespace X3D
  {
    namespace ECMA
    {
      struct Engine: Script::Engine
      {
        /**
         * \param sourceLine -1 for unknown.
         */
        Engine(Script *containingNode, string sourceCode,
               string sourceName, int sourceLine);
        ~Engine();

        struct ValueConverterBase
        {
          virtual bool fieldWrite(JSContext *, jsval, Node *,
                                  const FieldBase *, Time) const = 0;
          virtual void fieldRead(Node *, const FieldBase *,
                                 JSContext *, jsval *) const = 0;
        };

        Script *const containingNode;

        string getSourceName() const { return sourceName; }

        struct Field
        {
          const ValueConverterBase *valueConverter;
          jsval handler;
          Field(const ValueConverterBase *valueConverter):
            valueConverter(valueConverter), handler(JSVAL_NULL) {}
        };

        Field *getField(int i) { return &fields[i]; }

      private:
        string sourceName;
```

```
        JSRuntime *spiderMonkeyRuntime;
        JSContext *spiderMonkeyContext;

        vector<Field> fields;

        static JSBool getCustomField(JSContext *, JSObject *, jsval, jsval
              *);
        static JSBool setCustomField(JSContext *, JSObject *, jsval, jsval
              *);

        void handleEvent(int fieldIndex, Time timestamp);
      };
    }
  }
}

#endif // ARCHON_X3D_SCRIPT_ECMA_H
```

Listing: **archon/x3d/server/script_ecma.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

/*
 * \file script_ecma.C
 *
 * \todo We might need to use the special unicode variants of the
 * SpiderMonkey API calls. It seems though that it handles UTF8 in
 * strings with no problems (thanks to the cool features of the UTF8
 * encoding). Never the less, using UTF8 encoded identifiers makes it
 * barf. So if the X3D specification requires that a script input
 * field may use non-ascii characters then we need to switch to the
 * unicode aware API of SpiderMonkey.
 *
 * \todo Currently, events generated by the script methods will be
 * delivered imediately. This is against the X3D specification. See
 * X3D Specification, ISO/IEC 19775:200x 4.8.3.3 Responding to events
 * (3rd paragraph). The requrement of the specification, that events
 * generated in the course of evaluating a script method should be
 * buffered and delivered no earlier than when the method execution
 * completes, is sensible, since it eliminates potential problem of
 * reentrant script method invocations. It is proberbly best done by
```



```
 * simply binding each defined property to a jsval in the Invocation
 * structure and queued in the order of definition. Then upon
 * completion of the handler the queue could be processed.
 *
 * \todo Currently we do not properly support inputOutput fields since
 * we cannot have both a method and a property of the same name. How
 * is this to solved?
 *
 * \todo Currently we will not be able to handle a situation where a
 * handler for an inputOutput field writes to the same field. This
 * will cause an infinit recursion. See X3D Specification, ISO/IEC
 * 19775:200x 4.8.3.8 inputOutput fields and the containing node (2nd
 * paragraph).
 */

#include <archon/x3d/server/field_type.H>
#include <archon/x3d/server/dump.H>
#include <archon/x3d/server/script_ecma.H>
#include <archon/x3d/server/server.H>

namespace Archon
{
  namespace X3D
  {
    namespace ECMA
    {
      namespace
      {
        void printTypeOfJsval(JSContext *c, jsval v)
        {
          switch(JS_TypeOfValue(c, v))
          {
          case JSTYPE_VOID: cerr << "VOID\n"; break;
          case JSTYPE_OBJECT: cerr << "OBJECT\n"; break;
          case JSTYPE_FUNCTION: cerr << "FUNCTION\n"; break;
          case JSTYPE_STRING: cerr << "STRING\n"; break;
          case JSTYPE_NUMBER: cerr << "NUMBER\n"; break;
          case JSTYPE_BOOLEAN: cerr << "BOOLEAN\n"; break;
          case JSTYPE_LIMIT: cerr << "LIMIT\n"; break;
          }
        }

        /*
        JSBool global_myFunc(JSContext *context, JSObject *obj, uintN argc,
             jsval *argv, jsval *rval)
        {
          jsdouble x, y;

          if(!JS_ValueToNumber(context, argv[0], &x)) return JS_FALSE;
          y = fabs(x);
          cerr << "myFunc(" << x << ") = " << y << "\n";
          return JS_NewDoubleValue(context, y, rval);
        }
        */

        JSBool browser_print(JSContext *c, JSObject *o, uintN n, jsval *v,
             jsval *r)
        {
```

```
  for(unsigned i=0; i<n; ++i)
  {
    JSString *s = JS_ValueToString(c, v[i]);
    if(!s) return JS_FALSE;
    if(i) cerr << " ";
    cerr << JS_GetStringBytes(s);
  }
  return JS_TRUE;
}

JSFunctionSpec globalMethods[] =
{
  { "print", browser_print, 0 },
  { 0 }
};

JSFunctionSpec browserMethods[] =
{
  { "print", browser_print, 0 },
  { 0 }
};

JSBool globalResolveOp(JSContext *, JSObject *, jsval);

JSClass globalClass =
{
  "Global", 0,
  JS_PropertyStub,  JS_PropertyStub,JS_PropertyStub, JS_PropertyStub,
  JS_EnumerateStub, /*globalResolveOp*/JS_ResolveStub, JS_ConvertStub
       ,  JS_FinalizeStub
};

JSClass browserClass =
{
  "Browser", 0,
  JS_PropertyStub,  JS_PropertyStub,JS_PropertyStub, JS_PropertyStub,
  JS_EnumerateStub, JS_ResolveStub, JS_ConvertStub,  JS_FinalizeStub
};


void toJsval(JSContext *, bool v, jsval *r)
{
  *r = v ? JSVAL_TRUE : JSVAL_FALSE;
}

bool fromJsval(JSContext *c, jsval v, bool *r)
{
  JSBool w;
  if(!JS_ValueToBoolean(c, v, &w)) return false;
  *r = w;
  return true;
}


void toJsval(JSContext *c, double v, jsval *r)
{
  if(!JS_NewDoubleValue(c, v, r))
    ARCHON_THROW1(InternalException,
```



```
                  "ECMA::toJsval: Allocation error");
}

bool fromJsval(JSContext *c, jsval v, double *r)
{
  jsdouble w;
  if(!JS_ValueToNumber(c, v, &w)) return false;
  *r = w;
  return true;
}


void toJsval(JSContext *c, int v, jsval *r)
{
  if(!JS_NewDoubleValue(c, v, r))
    ARCHON_THROW1(InternalException,
                  "ECMA::toJsval: Allocation error");
}

bool fromJsval(JSContext *c, jsval v, int *r)
{
  if(!JS_ValueToInt32(c, v, r)) return false;
  return true;
}


void toJsval(JSContext *c, string v, jsval *r)
{
  JSString *s = JS_NewStringCopyZ(c, v.c_str());
  if(!s) ARCHON_THROW1(InternalException,
                       "ECMA::toJsval: Allocation error");
  *r = STRING_TO_JSVAL(s);
}

bool fromJsval(JSContext *c, jsval v, string *r)
{
  JSString *s =
    JSVAL_IS_STRING(v) ? JSVAL_TO_STRING(v) : JS_ValueToString(c, v);
  if(!s) ARCHON_THROW1(InternalException,
                       "ECMA::fromJsval: Allocation error");
  *r = string(JS_GetStringBytes(s));
  return true;
}


void toJsval(JSContext *c, Time v, jsval *r)
{
  long double w;
  v.get(w);
  if(!JS_NewDoubleValue(c, w, r))
    ARCHON_THROW1(InternalException,
                  "ECMA::toJsval: Allocation error");
}

bool fromJsval(JSContext *c, jsval v, Time *r)
{
  jsdouble w;
  if(!JS_ValueToNumber(c, v, &w)) return false;
```

```
  *r = Time(w);
  return true;
}



template<typename T>
struct SimpleValueConverter: Engine::ValueConverterBase
{
  bool fieldWrite(JSContext *c, jsval v, Node *n,
                  const FieldBase *f, Time t) const
  {
    T w;
    if(!fromJsval(c, v, &w)) return false;
    Ref<SimpleValue<T> > u = new SimpleValue<T>(f->getType(), w);
    Event e(u.get(), t);
    f->handleEvent(n, &e, true);
    return true;
  }

  void fieldRead(Node *n, const FieldBase *f,
                 JSContext *c, jsval *r) const
  {
    Ref<ValueBase> v = f->get(n);
    const SimpleValue<T> *w =
      dynamic_cast<const SimpleValue<T> *>(v.get());
    if(!w) ARCHON_THROW1(InternalException, "Illegal value type");
    toJsval(c, w->value, r);
  }
};

struct NodeContext
{
  Ref<Node> n;
  vector<const FieldBase *> fields;
  NodeContext(Ref<Node> n): n(n) {}
};

JSBool nodeResolveOp(JSContext *, JSObject *, jsval);
void nodeFinalizeOp(JSContext *, JSObject *);

JSClass nodeClass =
{
  "Node", JSCLASS_HAS_PRIVATE,
  JS_PropertyStub, JS_PropertyStub, JS_PropertyStub, JS_PropertyStub,
  JS_EnumerateStub, nodeResolveOp, JS_ConvertStub, nodeFinalizeOp
};

struct NodeValueConverter: Engine::ValueConverterBase
{
  bool fieldWrite(JSContext *c, jsval v, Node *n,
                  const FieldBase *f, Time t) const
  {
    Ref<Node> w;
    if(JSVAL_IS_NULL(v));
    else if(JSVAL_IS_OBJECT(v))
    {
      JSObject *o = JSVAL_TO_OBJECT(v);
```



```
      if(!JS_InstanceOf(c, o, &nodeClass, 0)) return false;
      NodeContext *nc = static_cast<NodeContext *>(JS_GetPrivate(c, o
          ));
      if(!nc)
        ARCHON_THROW1(InternalException,
                      "ECMA::NodeValueConverter::fieldWrite: "
                      "Node object had no context");
      w = nc->n;
    }
    else return false;
    Ref<NodeValue> u = new NodeValue(SFNode::type, w.get());
    Event e(u.get(), t);
    f->handleEvent(n, &e, true);
    return true;
  }


  /**
   * \todo We should consider caching the generated objects
   * for each field. Especially because we spend time defining
   * properties for it, and they will be lost for each new
   * 'field read'.
   */
  void fieldRead(Node *n, const FieldBase *f,
                 JSContext *c, jsval *r) const
  {
    Ref<ValueBase> v = f->get(n);
    const NodeValue *w = dynamic_cast<const NodeValue *>(v.get());
    if(!w) ARCHON_THROW1(InternalException, "Illegal value type");
    if(!w->value)
    {
      *r = JSVAL_NULL;
      return;
    }
    JSObject *o = JS_NewObject(c, &nodeClass, 0, 0);
    if(!o)
      ARCHON_THROW1(InternalException,
                    "ECMA::NodeValueConverter::fieldRead: "
                    "Could not create ECMA object");
    NodeContext *nc = new NodeContext(w->value);
    if(!JS_SetPrivate(c, o, nc))
    {
      delete nc;
      ARCHON_THROW1(InternalException,
                    "ECMA::NodeValueConverter::fieldRead: "
                    "Could not set private object data");
    }
    *r = OBJECT_TO_JSVAL(o);
  }
};

struct ValueConverterMap
{
  vector<const Engine::ValueConverterBase *> m;
  ValueConverterMap()
  {
    m.resize(fieldTypeIndexRoof, 0);
```

```
m[SFBool::index]      = new SimpleValueConverter<bool>;
//m[MFBool::index]      = new SimpleValueConverter<vector<bool
     > >;

/*
m[SFColor::index]     = new SimpleValueConverter<Vector3>;
m[MFColor::index]     = new SimpleValueConverter<vector<Vector3
     > >;

m[SFColorRGBA::index] = new SimpleValueConverter<Vector4>;
m[MFColorRGBA::index] = new SimpleValueConverter<vector<Vector4
     > >;
*/

m[SFDouble::index]    = new SimpleValueConverter<double>;
//m[MFDouble::index]    = new SimpleValueConverter<vector<double
     > >;

m[SFFloat::index]     = new SimpleValueConverter<double>;
//m[MFFloat::index]     = new SimpleValueConverter<vector<double
     > >;

/*
m[SFImage::index]     = new SimpleValueConverter<Image>;
m[MFImage::index]     = new SimpleValueConverter<vector<Image> >;
*/

m[SFInt32::index]     = new SimpleValueConverter<int>;
//m[MFInt32::index]     = new SimpleValueConverter<vector<int> >;

m[SFNode::index]      = new NodeValueConverter;
//m[MFInt32::index]     = new SimpleValueConverter<vector<int> >;

/*
m[SFRotation::index]  = new SimpleValueConverter<AxisRot3>;
m[MFRotation::index]  = new SimpleValueConverter<vector<AxisRot3
     > >;
*/

m[SFString::index]    = new SimpleValueConverter<string>;
//m[MFString::index]    = new SimpleValueConverter<vector<string
     > >;

m[SFTime::index]      = new SimpleValueConverter<Time>;
//m[MFTime::index]      = new SimpleValueConverter<vector<Time
     > >;

/*
m[SFVec2d::index]     = new SimpleValueConverter<Vector2>;
m[MFVec2d::index]     = new SimpleValueConverter<vector<Vector2
     > >;

m[SFVec2f::index]     = new SimpleValueConverter<Vector2>;
m[MFVec2f::index]     = new SimpleValueConverter<vector<Vector2
     > >;

m[SFVec3d::index]     = new SimpleValueConverter<Vector3>;
```



```
    m[MFVec3d::index]     = new SimpleValueConverter<vector<Vector3
         > >;

    m[SFVec3f::index]     = new SimpleValueConverter<Vector3>;
    m[MFVec3f::index]     = new SimpleValueConverter<vector<Vector3
         > >;
    */
  }
};

const Engine::ValueConverterBase *getValueConverter(const FieldType *
     t)
{
  static ValueConverterMap m;
  return m.m[t->getIndex()];
}



struct Invocation
{
  Engine *engine;
  Time timestamp;
  Invocation(Engine *engine, Time timestamp):
    engine(engine), timestamp(timestamp) {}
};

JSBool sfColorConstructOp(JSContext *c, JSObject *o, uintN n, jsval *
     v, jsval *r)
{
  cerr << "CONSTRUCT(" << n << ")\n";
  return JS_TRUE;
}

JSClass sfColorClass =
{
  "SFColor", 0,
  JS_PropertyStub, JS_PropertyStub, JS_PropertyStub, JS_PropertyStub,
  JS_EnumerateStub, JS_ResolveStub, JS_ConvertStub, JS_FinalizeStub,
  0, 0, 0, sfColorConstructOp
};

JSBool globalResolveOp(JSContext *c, JSObject *o, jsval i)
{
  if(!JSVAL_IS_STRING(i))
    ARCHON_THROW1(InternalException,
                  "ECMA::nodeResolveOp: "
                  "property ID was not a string");
  string n = JS_GetStringBytes(JSVAL_TO_STRING(i));
  cerr << "(" << n << ")\n";
  if(n == "SFColor")
  {
    //      JSObject *o2 =
      JS_DefineObject(c, o, "SFColor", &sfColorClass, 0,
           JSPROP_ENUMERATE);
    //JS_DefineFunctions(spiderMonkeyContext, browserObject,
         browserMethods);
  }
```

```
  return true;
}



Ref<CustomFieldBase> fetchCustomField(JSContext *c, jsval i,
                                      Engine::Field **f = 0,
                                      Script **s = 0,
                                      Time *t = 0)
{
  const Invocation *invocation =
    static_cast<const Invocation *>(JS_GetContextPrivate(c));
  if(!invocation)
    ARCHON_THROW1(InternalException,
                  "ECMA::fetchCustomField: "
                  "No bound invocation");
  Engine *engine = invocation->engine;
  Script *containingNode = engine->containingNode;
  if(!JSVAL_IS_INT(i))
    ARCHON_THROW1(InternalException,
                  "ECMA::fetchCustomField: "
                  "Unexpected type of property ID");
  int j = JSVAL_TO_INT(i);
  int n = containingNode->getNumberOfCustomFields();
  if(j<0 || j>n)
    ARCHON_THROW1(InternalException,
                  "ECMA::fetchCustomField: "
                  "Property ID out of range");
  if(f) *f = engine->getField(j);
  if(s) *s = containingNode;
  if(t) *t = invocation->timestamp;
  return containingNode->getCustomFieldNoLock(j);
}

void errorReporter(JSContext *context, const char *message,
                   JSErrorReport *report)
{
  const Invocation *invocation =
    static_cast<const Invocation *>(JS_GetContextPrivate(context));
  if(!invocation)
    ARCHON_THROW1(InternalException,
                  "ECMA::errorReporter: "
                  "No bound invocation");
  Logger *logger =
    invocation->engine->containingNode->context->server->getLogger();
  if(report)
    logger->log(invocation->engine->getSourceName() + ":" +
                Text::toString(report->lineno) + ": " + message);
  else
    logger->log(invocation->engine->getSourceName() + ": " + message
         );
}

JSBool setPropertyNo(JSContext *c, JSObject *, jsval i, jsval *)
{
  Ref<CustomFieldBase> f = fetchCustomField(c, i);
  string m = "AccessError: " + f->getName() + " is not writable";
  JS_ReportError(c, "%s", m.c_str());
```



```
  return JS_FALSE;
}

JSBool getPropertyNo(JSContext *c, JSObject *, jsval i, jsval *)
{
  Ref<CustomFieldBase> f = fetchCustomField(c, i);
  string m = "Access error: " + f->getName() + " is not readable";
  JS_ReportError(c, "%s", m.c_str());
  return JS_FALSE;
}



const FieldBase *fetchNodeField(JSContext *c, JSObject *o,
                                jsval i, Node **n)
{
  NodeContext *nc = static_cast<NodeContext *>(JS_GetPrivate(c, o));
  if(!nc)
    ARCHON_THROW1(InternalException,
                  "ECMA::fetchNodeField: "
                  "Object had no context");
  if(!JSVAL_IS_INT(i))
    ARCHON_THROW1(InternalException,
                  "ECMA::fetchNodeField: "
                  "Unexpected type of property ID");
  int j = JSVAL_TO_INT(i);
  if(j < 0 || unsigned(j) > nc->fields.size())
    ARCHON_THROW1(InternalException,
                  "ECMA::fetchNodeField: "
                  "Property ID out of range");
  if(n) *n = nc->n.get();
  return nc->fields[j];
}

JSBool getNodeField(JSContext *c, JSObject *o, jsval i, jsval *v)
{
  Node *n;
  const FieldBase *f = fetchNodeField(c, o, i, &n);
  getValueConverter(f->getType())->fieldRead(n, f, c, v);
  return JS_TRUE;
}

JSBool setNodeField(JSContext *c, JSObject *o, jsval i, jsval *v)
{
  const Invocation *invocation =
    static_cast<const Invocation *>(JS_GetContextPrivate(c));
  if(!invocation)
    ARCHON_THROW1(InternalException,
                  "ECMA::setNodeField: "
                  "No bound invocation");
  Node *n;
  const FieldBase *f = fetchNodeField(c, o, i, &n);
  return getValueConverter(f->getType())->
    fieldWrite(c, *v, n, f, invocation->timestamp) ?
    JS_TRUE : JS_FALSE;
}
```

```
  /**
   * \todo The access control cannot be done here because it
   * depends on the realization state of the node. It must be
   * done in the getters/setters. Also the currently used
   * 'getPropertyNo' and 'setPropertyNo' do the wrong thing.
   */
  JSBool nodeResolveOp(JSContext *c, JSObject *o, jsval i)
  {
    const Invocation *invocation =
      static_cast<const Invocation *>(JS_GetContextPrivate(c));
    if(!invocation)
      ARCHON_THROW1(InternalException,
                    "ECMA::nodeResolveOp: "
                    "No bound invocation");
    NodeContext *nc = static_cast<NodeContext *>(JS_GetPrivate(c, o));
    if(!nc)
      ARCHON_THROW1(InternalException,
                    "ECMA::nodeResolveOp: "
                    "Object had no context");
    if(!JSVAL_IS_STRING(i))
      ARCHON_THROW1(InternalException,
                    "ECMA::nodeResolveOp: "
                    "property ID was not a string");
    string n = JS_GetStringBytes(JSVAL_TO_STRING(i));
    const FieldBase *f = nc->n->getType()->lookupField(n);
    if(!f) return true;
    int j = nc->fields.size();
    if(j==255) ARCHON_THROW1(InternalException,
                             "ECMA::nodeResolveOp: "
                             "Too many fields in node");
    bool writable =
      invocation->engine->containingNode->getDirectOutput() &&
      f->getIsEventTarget();
    bool readable = f->getIsEventSource();
    if(!JS_DefinePropertyWithTinyId(c, o, n.c_str(), j,
                                    JSVAL_NULL,
                                    readable ? getNodeField :
                                         getPropertyNo,
                                    writable ? setNodeField :
                                         setPropertyNo,
                                    JSPROP_ENUMERATE|JSPROP_PERMANENT))
      ARCHON_THROW1(InternalException,
                    "ECMA::nodeResolveOp: "
                    "Could not define field as global property");
    nc->fields.push_back(f);
    return true;
  }

  void nodeFinalizeOp(JSContext *c, JSObject *o)
  {
    NodeContext *nc = static_cast<NodeContext *>(JS_GetPrivate(c, o));
    if(!nc)
      ARCHON_THROW1(InternalException,
                    "ECMA::nodeFinalizeOp: "
                    "Object had no context");
    delete nc;
  }
}
```



```
JSBool Engine::getCustomField(JSContext *c, JSObject *, jsval i, jsval
     *v)
{
  Field *g;
  Script *s;
  Ref<CustomFieldBase> f = fetchCustomField(c, i, &g, &s);
  g->valueConverter->fieldRead(s, f.get(), c, v);
  return JS_TRUE;
}

JSBool Engine::setCustomField(JSContext *c, JSObject *, jsval i, jsval
     *v)
{
  Field *g;
  Script *s;
  Time t;
  Ref<CustomFieldBase> f = fetchCustomField(c, i, &g, &s, &t);
  return g->valueConverter->
    fieldWrite(c, *v, s, f.get(), t) ? JS_TRUE : JS_FALSE;
}

Engine::Engine(Script *containingNode, string sourceCode,
               string sourceName, int sourceLine):
  containingNode(containingNode),
  sourceName(sourceName)
{
  // Construct an engine
  spiderMonkeyRuntime = JS_Init(1000000L);
  if(!spiderMonkeyRuntime)
    ARCHON_THROW1(InternalException,
                  "ECMA::Engine::Engine: Can't create "
                  "SpiderMonkey runtime");

  spiderMonkeyContext = JS_NewContext(spiderMonkeyRuntime, 8192);
  if(!spiderMonkeyContext)
    ARCHON_THROW1(InternalException,
                  "ECMA::Engine::Engine: Can't create "
                  "SpiderMonkey context");

  // Bind to the context
  Invocation invocation(this, containingNode->
                        context->server->getNextTimeStamp());
  JS_SetContextPrivate(spiderMonkeyContext, &invocation);

  JS_SetErrorReporter(spiderMonkeyContext, errorReporter);

  // Define standard ECMA classes
  JSObject *globalObject =
    JS_NewObject(spiderMonkeyContext, &globalClass, 0, 0);
  JS_InitStandardClasses(spiderMonkeyContext, globalObject);
  JS_DefineFunctions(spiderMonkeyContext, globalObject, globalMethods);

  // Define X3D SAI classes
  JSObject *browserObject =
    JS_DefineObject(spiderMonkeyContext, globalObject, "Browser",
                    &browserClass, 0, JSPROP_ENUMERATE);
```

```
JS_DefineFunctions(spiderMonkeyContext, browserObject, browserMethods
     );

// Define custom Script fields
unsigned n = containingNode->getNumberOfCustomFields();
if(n>255) ARCHON_THROW1(InternalException,
                        "ECMA::Engine::Engine: "
                        "Too many custom fields");
fields.reserve(n);
for(unsigned i=0; i<n; ++i)
{
  Ref<CustomFieldBase> f = containingNode->getCustomFieldNoLock(i);
  bool writable = f->getIsEventSource() || !f->getIsEventTarget();
  bool readable = !f->getIsEventSource() || f->getIsEventTarget();
  if(!JS_DefinePropertyWithTinyId(spiderMonkeyContext,
                                  globalObject, f->getName().c_str(),
                                  i, JSVAL_NULL,
                                  readable ? getCustomField :
                                       getPropertyNo,
                                  writable ? setCustomField :
                                       setPropertyNo,
                                  JSPROP_ENUMERATE|JSPROP_PERMANENT))
    ARCHON_THROW1(InternalException,
                  "ECMA::Engine::Engine: "
                  "Could not define field as global property");
  fields.push_back(Field(getValueConverter(f->getType())));
}

// Evaluate the script
jsval v;
uintN lineno = sourceLine < 1 ? 1 : sourceLine;
JS_EvaluateScript(spiderMonkeyContext, globalObject,
                  sourceCode.c_str(), sourceCode.size(),
                  sourceName.c_str(), lineno, &v);
/*
  cerr << "ECMA Engine created from " << sourceName << ":" <<
       sourceLine << "\n";
else cerr << "ECMA Engine failed to evaluate script from " <<
     sourceName << ":" << sourceLine << "\n";
*/

// Map readable fields to methods
for(unsigned i=0; i<n; ++i)
{
  Ref<CustomFieldBase> f = containingNode->getCustomFieldNoLock(i);
  if(f->getIsEventSource() && !f->getIsEventTarget()) continue;
  if(!JS_LookupProperty(spiderMonkeyContext,
                        globalObject, f->getName().c_str(), &v))
    ARCHON_THROW1(InternalException,
                  "ECMA::Engine::Engine: "
                  "JS_LookupProperty failed");
  if(JS_TypeOfValue(spiderMonkeyContext, v) != JSTYPE_FUNCTION)
    continue;
  jsval &h = fields[i].handler;
  h = v;
  /*
   * NOTE: It is very important that the 'fields' vector is
   * not resized from this point on as we store the address to
```



```
     * its elements with the SpiderMonkey garbage collector.
     */
    JS_AddRoot(spiderMonkeyContext, &h);
  }

  // Check for the existance of an 'initialize' method
  if(!JS_LookupProperty(spiderMonkeyContext,
                        globalObject, "initialize", &v))
    ARCHON_THROW1(InternalException,
                  "ECMA::Engine::Engine: "
                  "JS_LookupProperty failed");
  if(JS_TypeOfValue(spiderMonkeyContext, v) == JSTYPE_FUNCTION)
  {
    jsval r;
    JS_CallFunctionValue(spiderMonkeyContext, globalObject, v, 0, 0, &r
         );
  }

  // Dissociate from the context
  JS_SetContextPrivate(spiderMonkeyContext, 0);
}

Engine::~Engine()
{{ // The extra scope is needed to work around gcc3.2 bug #8287
  for(unsigned i=0; i<containingNode->getNumberOfCustomFields(); ++i)
  {
    jsval &h = fields[i].handler;
    if(JSVAL_IS_NULL(h)) continue;
    JS_RemoveRoot(spiderMonkeyContext, &h);
  }

  JS_DestroyContext(spiderMonkeyContext);
  JS_Finish(spiderMonkeyRuntime);
  //cerr << "ECMA Engine destroyed\n";
}}

/**
 * \todo Should we search for all the field methods during the
 * Engine initialization and then cache them as values for
 * faster access?
 */
void Engine::handleEvent(int i, Time timestamp)
{
  // Bind to the context
  Invocation invocation(this, containingNode->
                        context->server->getNextTimeStamp());
  JS_SetContextPrivate(spiderMonkeyContext, &invocation);

  int n = fields.size();
  if(i<0 || i>n)
    ARCHON_THROW1(InternalException,
                  "ECMA::Engine::handleEvent: "
                  "fieldIndex out of range");
  jsval handler = fields[i].handler;
  if(JSVAL_IS_NULL(handler)) return;
  Ref<CustomFieldBase> f = containingNode->getCustomFieldNoLock(i);
  jsval v[2];
  fields[i].valueConverter->fieldRead
```

```
          (containingNode, f.get(), spiderMonkeyContext, &v[0]);
        long double t1;
        timestamp.get(t1);
        jsdouble t2 = t1;
        if(!JS_NewDoubleValue(spiderMonkeyContext, t2, &v[1]))
          ARCHON_THROW1(InternalException,
                        "ECMA::Engine::handleEvent: "
                        "Allocation error");
        JSObject *globalObject = JS_GetGlobalObject(spiderMonkeyContext);
        jsval r;
        JS_CallFunctionValue(spiderMonkeyContext,
                             globalObject, handler, 2, v, &r);

        // Dissociate from the context
        JS_SetContextPrivate(spiderMonkeyContext, 0);
      }
    }
  }
}
```

Listing: **archon/x3d/server/server.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_SERVER_H
#define ARCHON_X3D_SERVER_H

#include <list>
#include <string>

#include <archon/util/logger.H>
#include <archon/util/mutex.H>
#include <archon/util/web_client.H>

#include <archon/x3d/server/scene.H>

namespace Archon
{
  namespace X3D
  {
    using namespace std;
```



```
    using namespace Utilities;

    // struct FontServer;

    struct Server: virtual BackRefObjectBase
    {
      struct Listener: virtual BackRefObjectBase {};

      Server(string name, double frameRate, Logger *);

      ~Server();

      string getName() const;
      Logger *getLogger() const;

      void setLoader(Ref<Loader>);
      Ref<Loader> getLoader() const;

      /*
      void setFontServer(Ref<FontServer>);
      Ref<FontServer> getFontServer() const;
      */

      void setListener(Ref<Listener>);
      Ref<Listener> getListener() const;

      void setRootContext(Ref<ExecutionContext>);
      Ref<ExecutionContext> getRootContext() const;

      double getFrameRate() const { return frameRate; }

      void print(string m) { logger->log(m); }

      /**
       * Requires scenegraph read lock.
       */
      Ref<ExecutionContext> getRootContextNoLock() { return rootContext; }

      void setProgressiveLoad(bool);
      bool getProgressiveLoad() const;

      string dumpScenegraph();

      Time getNextTimeStamp();

      /**
       * Activate time sensors.
       * Requires scenegraph write lock.
       */
      void tick();

      /**
       * This mutex protects the following parts of the scenegraph and
       * nothing else:
       *
       * - The value of every X3D field of every realized X3D Node
       * including the associated time stamps. Not the route lists
       * (EventSource)
```

```
   *
   * - The value of every custom field of a realized
   * CustomFileNode such as Script including the associated time
   * stamps. Not the route lists (EventSource)
   *
   * - The Engine object of a realized script node and the engine
   * reference of the containing node.
   *
   * - The root group reference variable in any realized
   * ExecutionContext. An ExecutionContext is realized when a
   * reference to it is added to a SubContextNode. This means that
   * any ExecutionContext that is reachable directly or indirectly
   * through the root scene along forward references is realized.
   *
   * - The 'drawCache' variable of realized ShapeNodes and any
   * ShapeCache object that is refered to by such a
   * variable. Also, the list of disposed shape caches
   * 'ShapeCache::getDisposedShapeCaches' is included here.
   *
   *
   * This is not an ideal situation. We would love to replace it
   * ith a more finegrained locking strategy, but so far we have
   * not come up with a good idea :-(
   *
   * One thing that seems to complicate matters is that rendering
   * needs to present a consistent scene graph, and never one that
   * is partially updated. What partially updated means, is not
   * absolutely clear to me (at the time of writing this), but as
   * I understand it, an event cascade needs to be carried out as
   * an atomic operation. Also when the SAI uses the buffered
   * update mode, then all changes has to be carried out as an
   * atomic operation with respect to rendering. That is, we
   * should not allow a frame to be rendered wich will show some
   * of the chages of such an update but not others.
   */
  const Mutex scenegraphMutex;

private:
  void refForwardDestroy();

  Mutex mutex;
  string name;
  Logger *logger;
  Ref<Loader> loader;
  //Ref<FontServer> fontServer;
  Ref<Listener> listener;
  Ref<ExecutionContext> rootContext;
  bool progressiveLoad;
  Time previousTimeStamp;
  double frameRate;

  friend struct ExecutionContext;
  /**
   * We need to use a different mutex to protect the 'contexts'
   * variable since this method is called from the Context
   * destructurs wich are called at times that are hard to
   * predict. Especially it is hard to avoid calling any Context
   * detructor while holding 'mutex'.
```



```
         */
        Mutex contextsMutex;
        list<ExecutionContext *> contexts;
        void add(ExecutionContext *);
        void remove(ExecutionContext *);
      };
    }
}

#endif // ARCHON_X3D_SERVER_H
```

Listing: **archon/x3d/server/server.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#include <archon/x3d/server/server.H>

//#include <archon/x3d/server/font.H>

namespace Archon
{
  namespace X3D
  {
    Server::Server(string name, double frameRate, Logger *logger):
      name(name), logger(logger), progressiveLoad(false), frameRate(frameRate
           )
    {
    }

    Server::~Server()
    {
      logger->log("No memory leak this time!");
    }

    void Server::refForwardDestroy()
    {
      //fontServer.reset();
      listener.reset();
      rootContext.reset();
      loader.reset(); // Should be last, since it waits for all jobs to
           complete
```

```
}

Ref<Loader> Server::getLoader() const
{
  Mutex::Lock l(mutex);
  return loader;
}

void Server::setLoader(Ref<Loader> v)
{
  Mutex::Lock l(mutex);
  loader = v;
}

/*
Ref<FontServer> Server::getFontServer() const
{
  Mutex::Lock l(mutex);
  return fontServer;
}

void Server::setFontServer(Ref<FontServer> v)
{
  Mutex::Lock l(mutex);
  fontServer = v;
}
*/

Ref<Server::Listener> Server::getListener() const
{
  Mutex::Lock l(mutex);
  return listener;
}

void Server::setListener(Ref<Listener> v)
{
  Mutex::Lock l(mutex);
  listener = v;
}

Ref<ExecutionContext> Server::getRootContext() const
{
  Mutex::Lock l(scenegraphMutex);
  return rootContext;
}

void Server::setRootContext(Ref<ExecutionContext> c)
{
  if(c) c->realize();
  Mutex::Lock l(scenegraphMutex);
  rootContext = c;
}

void Server::setProgressiveLoad(bool v)
{
  Mutex::Lock l(mutex);
  progressiveLoad = v;
}
```



```
bool Server::getProgressiveLoad() const
{
  Mutex::Lock l(mutex);
  return progressiveLoad;
}

string Server::getName() const
{
  Mutex::Lock l(mutex);
  return name;
}

Logger *Server::getLogger() const
{
  Mutex::Lock l(mutex);
  return logger;
}

string Server::dumpScenegraph()
{
  Mutex::Lock l(scenegraphMutex);
  return rootContext->dump();
}

Time Server::getNextTimeStamp()
{
  Mutex::Lock l(mutex);
  ++previousTimeStamp;
  Time t = Time::now();
  if(t > previousTimeStamp) previousTimeStamp = t;
  return previousTimeStamp;
}

void Server::add(ExecutionContext *c)
{
  Mutex::Lock l(contextsMutex);
  contexts.push_back(c);
}

void Server::remove(ExecutionContext *c)
{
  Mutex::Lock l(contextsMutex);
  contexts.remove(c);
}

void Server::tick()
{
  list<Ref<ExecutionContext> > t;
  {
    Mutex::Lock l(contextsMutex);
    list<ExecutionContext *>::iterator i = contexts.begin();
    while(i != contexts.end())
    {
      Ref<ExecutionContext> r(*i, RefSafeIncTag());
      if(r) t.push_back(r);
      ++i;
    }
```

```
        }
        {
          list<Ref<ExecutionContext> >::iterator i = t.begin();
          while(i != t.end())
          {
            (*i)->tick();
            ++i;
          }
        }
      }
    }
}
```

Listing: **archon/x3d/server/text.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_TEXT_H
#define ARCHON_X3D_TEXT_H

/*
 * \file text.H
 *
 *  X3D Text component
 */

#include <archon/x3d/server/geometry.H>

namespace Archon
{
  namespace X3D
  {
    void addFontPath(string);

    struct FontStyleNode: GeometricPropertyNode
    {
      static const NodeType *type;

    protected:
      FontStyleNode() {}
    };
```



```
    struct FontStyle: FontStyleNode
    {
      static Ref<Node> instantiate(BackRef<ExecutionContext> c)
      {
        return new FontStyle(c);
      }

      static const NodeType *type;
      const NodeType *getType() const { return type; }

      FontStyle(BackRef<ExecutionContext> c):
        Node(c),
        horizontal(true),
        leftToRight(true),
        size(1),
        spacing(1),
        topToBottom(true) {}

      const vector<string> &getFamily()      const { return family;      }
      bool                  getHorizontal()  const { return horizontal;  }
      const vector<string> &getJustify()     const { return justify;     }
      string                getLanguage()    const { return language;    }
      bool                  getLeftToRight() const { return leftToRight; }
      double                getSize()        const { return size;        }
      double                getSpacing()     const { return spacing;     }
      string                getStyle()       const { return style;       }
      bool                  getTopToBottom() const { return topToBottom; }

    private:
      friend void initializeTextComponent();

      vector<string> family;
      bool           horizontal;
      vector<string> justify;
      string         language;
      bool           leftToRight;
      double         size;
      double         spacing;
      string         style;
      bool           topToBottom;
    };

    struct TextNode: GeometryNode
    {
      static const NodeType *type;

    protected:
      TextNode() {}
    };

    struct Text: TextNode
    {
      static Ref<Node> instantiate(BackRef<ExecutionContext> c)
      {
        return new Text(c);
      }
```

```
      static const NodeType *type;
      const NodeType *getType() const { return type; }

      Text(BackRef<ExecutionContext> c):
        Node(c),
        maxExtent(0) {}

      vector<string>           getString()    const { return _string;   }
      Ref<const FontStyleNode> getFontStyle() const { return fontStyle; }
      vector<double>           getLength()    const { return length;    }
      double                   getMaxExtent() const { return maxExtent; }

      int intersect(const Math::Ray3 &, double &dist) const;
      void render(bool texture, bool color,
                  const Shape *shape,
                  const RenderConfig *) const;

    private:
      friend void initializeTextComponent();

      vector<string>     _string;
      Ref<FontStyleNode> fontStyle;
      vector<double>     length;
      double             maxExtent;

      EventSource stringChanged;
      EventSource fontStyleChanged;
      EventSource lengthChanged;
      EventSource maxExtentChanged;

      Time stringStamp;
      Time fontStyleStamp;
      Time lengthStamp;
      Time maxExtentStamp;
    };
  }
}

#endif // ARCHON_X3D_TEXT_H
```

Listing: **archon/x3d/server/text.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
```



```
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

/*
 * \file text.C
 *
 *  X3D Text component
 */

#include <iostream>

#include <ft2build.h>
#include FT_FREETYPE_H

#include <GL/gl.h>

#include <archon/util/file.H>
#include <archon/util/unicode.H>
#include <archon/util/image.H>

#include <archon/x3d/server/field_type.H>
#include <archon/x3d/server/field.H>
#include <archon/x3d/server/text.H>

namespace Archon
{
  namespace X3D
  {
    extern long nodeTypeIndexEnumerator;

    const NodeType *FontStyleNode::type = 0;
    const NodeType *FontStyle::type = 0;
    const NodeType *TextNode::type = 0;
    const NodeType *Text::type = 0;

    void initializeTextComponent()
    {
      vector<const FieldBase *> fields;

      // FontStyleNode

      FontStyleNode::type =
        NodeType::newAbstract("FontStyleNode", false, 0,
                              GeometricPropertyNode::type);


      // FontStyle

      fields.push_back(newPrivateField("family", MFString::type,
                                       &FontStyle::family));
      fields.push_back(newPrivateField("horizontal", SFBool::type,
                                       &FontStyle::horizontal));
      fields.push_back(newPrivateField("justify", MFString::type,
                                       &FontStyle::justify));
      fields.push_back(newPrivateField("language", SFString::type,
                                       &FontStyle::language));
      fields.push_back(newPrivateField("leftToRight", SFBool::type,
```

```
                                 &FontStyle::leftToRight));
  fields.push_back(newPrivateField("size", SFFloat::type,
                                 &FontStyle::size));
  fields.push_back(newPrivateField("spacing", SFFloat::type,
                                 &FontStyle::spacing));
  fields.push_back(newPrivateField("style", SFString::type,
                                 &FontStyle::style));
  fields.push_back(newPrivateField("topToBottom", SFBool::type,
                                 &FontStyle::topToBottom));
  FontStyle::type =
    NodeType::newConcrete("FontStyle", FontStyle::instantiate, &fields,
                          FontStyleNode::type);
  fields.clear();


  // TextNode

  TextNode::type =
    NodeType::newAbstract("TextNode", false, 0,
                          GeometryNode::type);


  // Text

  fields.push_back(newExposedField("string", MFString::type,
                                 &Text::_string,
                                 &Text::stringChanged,
                                 &Text::stringStamp));
  fields.push_back(newExposedField("fontStyle",
                                 &Text::fontStyle,
                                 &Text::fontStyleChanged,
                                 &Text::fontStyleStamp));
  fields.push_back(newExposedField("length", MFFloat::type,
                                 &Text::length,
                                 &Text::lengthChanged,
                                 &Text::lengthStamp));
  fields.push_back(newExposedField("maxExtent", SFFloat::type,
                                 &Text::maxExtent,
                                 &Text::maxExtentChanged,
                                 &Text::maxExtentStamp));
  Text::type =
    NodeType::newConcrete("Text", Text::instantiate, &fields,
                          TextNode::type);
  fields.clear();
}

/**
 * \todo The X3D specification says that "The Text node does not
 * participate in collision detection", but does this mean that
 * one cannot activate a pointing device sensor via the text node,
 * or does it only refer to the fact that the text node will not
 * interfear with your avatar when walking?
 */
int Text::intersect(const Math::Ray3 &, double &dist) const
{
  return 0;
}
```



```
namespace
{
  struct Glyph
  {
    /**
     * Size of layout bbox in units of standard line spacing.
     */
    Vector2 size;

    /**
     * Position of lower left corner of bbox of horizontal layout
     * in units of standard line spacing relative to the pen
     * position on the baseline.
     */
    Vector2 horizontalLowerLeft;

    /**
     * Pen advance for horizontal layout.
     */
    double horizontalAdvance;

    /**
     * Position of lower left corner of bbox of vertical layout in
     * units of standard line spacing relative to the pen position
     * on the baseline.
     */
    Vector2 verticalLowerLeft;

    /**
     * Pen advance for vertical layout.
     */
    double verticalAdvance;

    Vector2 textureLowerLeft;
    Vector2 textureUpperRight;

    Glyph(Vector2 size,
          Vector2 horizontalLowerLeft,
          double horizontalAdvance,
          Vector2 verticalUpperRight,
          double verticalAdvance,
          Vector2 textureLowerLeft,
          Vector2 textureUpperRight):
      size(size),
      horizontalLowerLeft(horizontalLowerLeft),
      horizontalAdvance(horizontalAdvance),
      verticalLowerLeft(verticalLowerLeft),
      verticalAdvance(verticalAdvance),
      textureLowerLeft(textureLowerLeft),
      textureUpperRight(textureUpperRight) {}
  };

  struct FontFace;

  struct FontList
  {
    Mutex mutex;
```

```
static vector<string> fontPaths;
static int pixelSize;

struct Entry
{
  string plainFilePath;
  int plainFileIndex;
  FontFace *plainFace;

  string boldFilePath;
  int boldFileIndex;
  FontFace *boldFace;

  string italicFilePath;
  int italicFileIndex;
  FontFace *italicFace;

  string bolditalicFilePath;
  int bolditalicFileIndex;
  FontFace *bolditalicFace;

  Entry():
    plainFace(0), boldFace(0),
    italicFace(0), bolditalicFace(0) {}
};

map<string, Entry> fontFamilies;

FT_Library library;

bool addFace(string path, int index, FT_Face face)
{
  string family = face->family_name;

  bool bold   = face->style_flags&FT_STYLE_FLAG_BOLD;
  bool italic = face->style_flags&FT_STYLE_FLAG_ITALIC;

  //string style = bold ? italic ? "bolditalic" : "bold" : italic ? "
       italic" : "plain";
  //cerr << "Found \"" << family << "\" " << style << "\n";

  /*
  cerr << "Style name       = " << face->style_name << "\n";
  cerr << "Number of glyphs = " << face->num_faces << "\n";
  cerr << "Units per EM     = " << face->units_per_EM << "\n";
  cerr << "Height           = " << face->height << "\n";
  cerr << "Ascender         = " << face->ascender << "\n";
  cerr << "Descender        = " << face->descender << "\n";
  cerr << "Num fixed sizes  = " << face->num_fixed_sizes << "\n";
  cerr << "Num charmaps     = " << face->num_charmaps << "\n";
  if(face->face_flags&FT_FACE_FLAG_SCALABLE)
    cerr << "                   FT_FACE_FLAG_SCALABLE\n";
  if(face->face_flags&FT_FACE_FLAG_FIXED_SIZES)
    cerr << "                   FT_FACE_FLAG_FIXED_SIZES\n";
  if(face->face_flags&FT_FACE_FLAG_FIXED_WIDTH)
    cerr << "                   FT_FACE_FLAG_FIXED_WIDTH\n";
  if(face->face_flags&FT_FACE_FLAG_HORIZONTAL)
```



```
    cerr << "                   FT_FACE_FLAG_HORIZONTAL\n";
  if(face->face_flags&FT_FACE_FLAG_VERTICAL)
    cerr << "                   FT_FACE_FLAG_VERTICAL\n";
  if(face->face_flags&FT_FACE_FLAG_SFNT)
    cerr << "                   FT_FACE_FLAG_SFNT\n";
  if(face->face_flags&FT_FACE_FLAG_KERNING)
    cerr << "                   FT_FACE_FLAG_KERNING\n";
  if(face->face_flags&FT_FACE_FLAG_MULTIPLE_MASTERS)
    cerr << "                   FT_FACE_FLAG_MULTIPLE_MASTERS\n";
  if(face->face_flags&FT_FACE_FLAG_GLYPH_NAMES)
    cerr << "                   FT_FACE_FLAG_GLYPH_NAMES\n";
  if(face->face_flags&FT_FACE_FLAG_EXTERNAL_STREAM)
    cerr << "                   FT_FACE_FLAG_EXTERNAL_STREAM\n";
  if(face->face_flags&FT_FACE_FLAG_FAST_GLYPHS)
    cerr << "                   FT_FACE_FLAG_FAST_GLYPHS\n";
  */

  Entry &entry = fontFamilies[family];

  if(!bold && !italic)
  {
    if(entry.plainFilePath.size()) return false;
    entry.plainFilePath = path;
    entry.plainFileIndex = index;
  }
  else if(bold && !italic)
  {
    if(entry.boldFilePath.size()) return false;
    entry.boldFilePath = path;
    entry.boldFileIndex = index;
  }
  else if(!bold && italic)
  {
    if(entry.italicFilePath.size()) return false;
    entry.italicFilePath = path;
    entry.italicFileIndex = index;
  }
  else if(bold && italic)
  {
    if(entry.bolditalicFilePath.size()) return false;
    entry.bolditalicFilePath = path;
    entry.bolditalicFileIndex = index;
  }

  return true;
}

FontList()
{
  if(FT_Init_FreeType(&library))
    ARCHON_THROW1(ResourceException,
                  "Error initializing FreeType library");

  int n = 0;

  for(unsigned i=0; i<fontPaths.size(); ++i)
  {
    //cerr << "Searching for fonts in \"" << fontPaths[i] << "\"\n";
```

```
      vector<string> dirNames = File::getDirNames(fontPaths[i]);
      for(unsigned j=0; j<dirNames.size(); ++j)
      {
        string name = dirNames[j];
        if(name == "." || name == "..") continue;
        string p = fontPaths[i]+"/"+name;
        if(File::Stat(p).getType() != File::Stat::type_regular)
             continue;

        FT_Face face;
        if(FT_New_Face(library, p.c_str(), 0, &face)) continue;

        //cerr << "Found font file \"" << p << "\"\n";

        if(addFace(p, 0, face)) ++n;

        unsigned m = face->num_faces;
        FT_Done_Face(face);

        for(unsigned k=1; k<m; ++k)
        {
          if(FT_New_Face(library, p.c_str(), k, &face)) break;
          if(addFace(p, k, face)) ++n;
          FT_Done_Face(face);
        }
      }
    }

    cerr << "Found " + Archon::Utilities::Text::toString(n) +
      " font faces in " + Archon::Utilities::Text::toString(fontPaths.
           size()) +
      " directories\n";
  }

  FontFace *getFace(string familyName, bool bold, bool italic);
  FontFace *getDefaultFace();
};

vector<string> FontList::fontPaths;
int FontList::pixelSize = 32;

struct FontFace
{
  Mutex mutex;

  double horizontalBottom;
  double horizontalTop;

  double verticalLeft;
  double verticalRight;

  vector<const Glyph *> glyphs;

  GLuint textureId;

  FontFace(string filePath, int faceIndex, int pixelSize)
  {
    const FontList *fontList = getFontList();
```



```
    FT_Face face;
    if(FT_New_Face(fontList->library, filePath.c_str(), faceIndex, &
         face))
      ARCHON_THROW1(ResourceException,
                    "Font file \"" + filePath + "\" is no longer
                         available");

    if(FT_Set_Pixel_Sizes(face, 0, pixelSize))
      ARCHON_THROW1(InternalException,
                    "Error setting pixel size of font face");

    FT_GlyphSlot slot = face->glyph;

    glyphs.resize(256);

    int imageWidth  = 16*pixelSize;
    int imageHeight = 16*pixelSize;
    Image texture(imageWidth, imageHeight, Image::components_la, 8);
         // Could be reduced to 1 bit per component

    for(int x=0; x<imageWidth; ++x)
      for(int y=0; y<imageHeight; ++y)
        texture.setPixel(x, y, 255u, 0u);

    /*
     * Minimum vertical and horizontal distance between glyphs
     * in texture mesured in number of texels. This spacing is
     * introduced to prevent the texels from one glyph to
     * "bleed" into the bbox of an adjacent glyph due to the
     * filtering done by OpenGL.
     */
    const int textureSpacing = 4;

    /*
     * Number of texels to expand the minimum bbox of the glyph
     * in the texture by to get the bbox to use for the actual
     * texture mapping. This of cause increases the size of the
     * rendered quad for the glyph. The reason for expanding the
     * minimum bbox is to allow the filtering done by OpenGL to
     * work correctly at the edges of the glyph. If there were
     * no padding then otherwise smooth edges of the glyph would
     * be cut off too sharply. Note that half the size of
     * 'textureSpacing' will usually be optimal.
     */
    const double texturePadding = textureSpacing/2.0;

    int xPos = textureSpacing;
    int yPos = textureSpacing;
    int maxHeight = 0;

    // Load first Unicode page into texture
    for(unsigned i=0; i<256; ++i)
    {
      if(FT_Load_Char(face, static_cast<unsigned char>(i),
           FT_LOAD_RENDER))
        continue;
      if(int(slot->bitmap.pixel_mode) != 2)
```

```
  ARCHON_THROW1(InternalException,
                "Unsupported pixel data encoding in font");

if(slot->bitmap.width+textureSpacing > imageWidth - xPos)
{
  yPos += maxHeight+textureSpacing;
  xPos = textureSpacing;
  maxHeight = 0;
}

if(slot->bitmap.width+textureSpacing > imageWidth - xPos ||
   slot->bitmap.rows+textureSpacing > imageHeight - yPos)
  ARCHON_THROW1(InternalException,
                "All glyphs won't fit inside texture");

for(int x=0; x<slot->bitmap.width; ++x)
  for(int y=0; y<slot->bitmap.rows; ++y)
  {
    unsigned l = (static_cast<unsigned char *>(slot->bitmap.
         buffer))
      [x + y * slot->bitmap.width];
    texture.setPixel(xPos+x, imageHeight-1-yPos-y, 255, l);
  }

double linesPerTexel = face->units_per_EM/double(face->height*
     pixelSize);

horizontalBottom = 0;
horizontalTop    = pixelSize*linesPerTexel;
verticalLeft     = 0;
verticalRight    = pixelSize*linesPerTexel;

Vector2 horizontalLowerLeft;
double horizontalAdvance;
Vector2 verticalLowerLeft;
double verticalAdvance;

if(face->face_flags&FT_FACE_FLAG_HORIZONTAL)
{
  horizontalLowerLeft = verticalLowerLeft =
    Vector2(slot->bitmap_left-texturePadding,
            slot->bitmap_top-slot->bitmap.rows-texturePadding)*
    linesPerTexel;

  horizontalAdvance = slot->advance.x/64 * linesPerTexel;
  verticalAdvance   = 1;
}
else
{
  horizontalLowerLeft = verticalLowerLeft =
    Vector2(slot->bitmap_left-texturePadding,
            slot->bitmap_top-slot->bitmap.rows-texturePadding)*
    linesPerTexel;

  horizontalAdvance = 1;
  verticalAdvance   = slot->advance.y/64 * linesPerTexel;
}
```



```
  glyphs[i] =
    new Glyph(Vector2(slot->bitmap.width + 2*texturePadding,
                      slot->bitmap.rows  + 2*texturePadding) *
              linesPerTexel,
              horizontalLowerLeft, horizontalAdvance,
              verticalLowerLeft, verticalAdvance,
              Vector2(double(xPos-texturePadding)/imageWidth,
                      double(imageHeight-1-yPos-slot->bitmap.rows-
                           texturePadding)/imageHeight),
              Vector2(double(xPos+slot->bitmap.width+texturePadding
                   )/imageWidth,
                      double(imageHeight-1-yPos+texturePadding)/
                           imageHeight));

  xPos += slot->bitmap.width+textureSpacing;
  if(slot->bitmap.rows > maxHeight) maxHeight = slot->bitmap.rows;
}

{
  string family = face->family_name;
  bool bold   = face->style_flags&FT_STYLE_FLAG_BOLD;
  bool italic = face->style_flags&FT_STYLE_FLAG_ITALIC;
  string style = bold ? italic ? "bolditalic" : "bold" : italic ? "
       italic" : "plain";
  texture.save(family + " " + style + ".png");
}

FT_Done_Face(face);

{
  int colorMode;
  switch(texture.getComponentSpecifier())
  {
  case Utilities::Image::components_l:    colorMode = GL_LUMINANCE
       ;       break;
  case Utilities::Image::components_la:   colorMode =
       GL_LUMINANCE_ALPHA; break;
  case Utilities::Image::components_rgb:  colorMode = GL_RGB;
       break;
  case Utilities::Image::components_rgba: colorMode = GL_RGBA;
       break;
  default:
    ARCHON_THROW1(InternalException,
                  "Unsuported color components");
  }

  int componentType;
  switch(texture.getBitsPerComponent())
  {
  case 8:  componentType = GL_UNSIGNED_BYTE;  break;
  case 16: componentType = GL_UNSIGNED_SHORT; break;
  default:
    ARCHON_THROW1(InternalException,
                  "Unsuported color component width");
  }

  glGenTextures(1, &textureId);
  glBindTexture(GL_TEXTURE_2D, textureId);
```

```
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,GL_CLAMP);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
      glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

      glTexImage2D(GL_TEXTURE_2D, 0, colorMode, texture.getWidth(),
                   texture.getHeight(), 0, colorMode, componentType,
                   texture.getDataPtr());
    }
  }

  /**
   * \todo Must dynamically detect missing unicode pages and
   * create them on demand
   */
  const Glyph *getGlyph(uchar i)
  {
    Mutex::Lock l(mutex);
    if(i > 255) return 0;
    return glyphs[i];
  }

  /**
   * Get the default face of the default font family.
   *
   * \return Never null.
   */
  static FontFace *get()
  {
    FontList *fontList = getFontList();
    return fontList->getDefaultFace();
  }

  /**
   * Get the specified face of the specified font family.
   *
   * \return Null if the font face was not found.
   */
  static FontFace *get(string familyName, bool bold, bool italic)
  {
    FontList *fontList = getFontList();
    return fontList->getFace(familyName, bold, italic);
  }

private:
  static FontList *getFontList()
  {
    static FontList fontList;
    return &fontList;
  }
};
}

FontFace *FontList::getDefaultFace()
{
  FontFace *f = getFace("Times New Roman", false, false);
  if(f) return f;
```



```
  ARCHON_THROW1(InternalException,
                "Could not find default font face");
}

FontFace *FontList::getFace(string familyName, bool bold, bool italic)
{
  Mutex::Lock l(mutex);

  map<string, Entry>::iterator i = fontFamilies.find(familyName);
  if(i == fontFamilies.end()) return 0;

  if(!bold && !italic)
  {
    if(!i->second.plainFilePath.size()) return 0;
    if(!i->second.plainFace)
      i->second.plainFace =
        new FontFace(i->second.plainFilePath, i->second.plainFileIndex,
             pixelSize);
    return i->second.plainFace;
  }

  if(bold && !italic)
  {
    if(!i->second.boldFilePath.size()) return 0;
    if(!i->second.boldFace)
      i->second.boldFace =
        new FontFace(i->second.boldFilePath, i->second.boldFileIndex,
             pixelSize);
    return i->second.boldFace;
  }

  if(!bold && italic)
  {
    if(!i->second.italicFilePath.size()) return 0;
    if(!i->second.italicFace)
      i->second.italicFace =
        new FontFace(i->second.italicFilePath, i->second.italicFileIndex
             , pixelSize);
    return i->second.italicFace;
  }

  if(!i->second.bolditalicFilePath.size()) return 0;
  if(!i->second.bolditalicFace)
    i->second.bolditalicFace =
      new FontFace(i->second.bolditalicFilePath, i->second.
           bolditalicFileIndex, pixelSize);
  return i->second.bolditalicFace;
}



/**
 * The meaning of a line spacing of 1 (as specified by the
 * FontStyle node) is taken to be relativly equal to the nominal
 * line spacing explicitly specified by the font.
 *
 * \todo Support vertical layout.
 *
```

```
 * \todo Support texturing through TextureNodes on text by
 * utilizing OpenGLs multi texture support.
 *
 * \todo Support more that the ISO Latin-1 subset of the Unicode
 * character set. This will require some sort op paging, ie. a
 * table of font pages the are currently available. A page could
 * obviously have size "256 glyphs", but smaller sizes may lead to
 * better memory utilization (less waste).
 *
 * \todo We must recognize the standard font family names "SERIF",
 * "SANS" and "TYPEWRITER" and be able to map them to specific
 * font faces that conform with the X3D specification. Currently
 * we just check for one alternative name before giving up. the
 * alternative name for "SERIF" is "Times New Roman", for "SANS"
 * it is "Arial" and for "TYPEWRITER" it is "Courier New". This
 * works fine if you have access to the fonts from a Microsoft
 * Windows installation. But a better alternative should be
 * strived for.
 *
 * \todo Font glyphs could be packed much better in the texture
 * than they are right now. Currently we waste a lot of precious
 * texture memory, and it quickly ruins performance.
 *
 * \todo Apply kerning information. See the FreeType separate
 * glyph documentation if you feel clueless.
 *
 * \todo Because the text rendering utilizes transparency for
 * anti-aliasing purposes, the order of rendering starts to get
 * important. Currently there is no control of rendering order,
 * that is, nodes are rendered in the order they occur in the
 * source file. One possible way to deal with this problem would
 * be to detect polygons with transparent features, and instead of
 * rendering them rigt away they should be saved with global
 * coordinates and all OpenGL properties into a BSP tree. Then
 * when all the opaque polygons are rendered the polygons from the
 * BSP tree are rendered in depth order with 'write to
 * depthbuffer' disabled. A drawback is that these polygons cannot
 * be compiled into a calllist as usual. Proberbly a better
 * alternative to store individual polygons in a BSP tree would be
 * the postpone rendering of the individual text nodes. This
 * should be done by somehow saving the local coordinate system
 * and information about which directional light sources that are
 * active. Then the text nodes can be sorted in order of distance
 * to the local origin and rendered in that order after the normal
 * rendering. This will not produce an optimal result with respect
 * to quality when text nodes are intersecting each or just too
 * close together, but it might be good enough. Another way to
 * deal with the problem might be to utilize the alpha buffer. Yet
 * another way would be to abandon the anti-aliasing feature and
 * then use alpha-testing to achive on/off-transparency. This can
 * be done without regard to rendering order. Finally we should
 * considder these problems togeter with the more general
 * challenge in getting transparent textures to work on any kind
 * of geometry.
 */
void Text::render(bool texture, bool color,
                  const Shape *shape,
                  const RenderConfig *) const
```



```
{
  const FontStyle *f = dynamic_cast<const FontStyle *>(fontStyle.get());

  if(!f && fontStyle)
    ARCHON_THROW1(InternalException,
                  "'" + fontStyle->getType()->getName() +
                  "' is not supported yet");

  FontFace *face = 0;

  if(f) for(unsigned i=0; i<f->getFamily().size(); ++i)
  {
    bool bold   = false;
    bool italic = false;
    if(f->getStyle() == "BOLD") bold = true;
    else if(f->getStyle() == "ITALIC") italic = true;
    else if(f->getStyle() == "BOLDITALIC") bold = italic = true;

    face = FontFace::get(f->getFamily()[i], bold, italic);
    if(face) break;

    if(f->getFamily()[i] == "SERIF")
    {
      face = FontFace::get("Times New Roman", bold, italic);
      if(face) break;
    }
    else if(f->getFamily()[i] == "SANS")
    {
      face = FontFace::get("Arial", bold, italic);
      if(face) break;
    }
    else if(f->getFamily()[i] == "TYPEWRITER")
    {
      face = FontFace::get("Courier New", bold, italic);
      if(face) break;
    }
  }

  if(!face) face = FontFace::get();

  bool horizontal  = f ? f->getHorizontal()  : true;
  bool leftToRight = f ? f->getLeftToRight() : true;
  bool topToBottom = f ? f->getTopToBottom() : true;

  const double size = f ? f->getSize() : 1;
  double lineAdvance = f ? f->getSpacing()*size : 1;
  if(topToBottom) lineAdvance = -lineAdvance;

  vector<vector<const Glyph *> > stringGlyphs;
  vector<double> stringExtents;
  vector<double> stringAjusts; // Inter character ajustment

  stringGlyphs.resize(_string.size());
  stringExtents.resize(_string.size());
  stringAjusts.resize(_string.size());

  // Find glyphs and calculate extents of strings
  double max = 0;
```

```
for(unsigned i=0; i<_string.size(); ++i)
{
  ustring s = Unicode::decodeUtf8(_string[i]);

  stringGlyphs[i].reserve(s.size());
  double extent = 0;

  for(unsigned j=0; j<s.size(); ++j)
  {
    const Glyph *g = face->getGlyph(s[j]);
    if(!g) continue;
    stringGlyphs[i].push_back(g);
    extent += (horizontal ? g->horizontalAdvance : g->verticalAdvance)*
         size;
  }

  if(i >= length.size() || length[i] == 0)
  {
    stringExtents[i] = extent;
    stringAjusts[i] = 0;
  }
  else
  {
    stringExtents[i] = length[i];
    if(stringGlyphs[i].size() > 1)
      stringAjusts[i] = (length[i] - extent)/(stringGlyphs[i].size
           ()-1);
  }

  if(stringExtents[i] > max) max = stringExtents[i];
}

double compress = maxExtent > 0 && max > maxExtent ? maxExtent/max : 1;

/*
 * The displacement of the lines in the primary direction of the
 * lines and relative the the effective length of each line.
 * before the origin of alignment
 */
double primaryAlignment = 0;
if(f && f->getJustify().size())
{
  string s = f->getJustify()[0];
  if(s == "MIDDLE") primaryAlignment = -0.5;
  else if(s == "END") primaryAlignment = -1;
}
const bool primaryDirection = horizontal ? leftToRight : topToBottom;
if(!primaryDirection) primaryAlignment = - primaryAlignment;

double totalSecondaryExtent =
  (horizontal ? face->horizontalTop - face->horizontalBottom :
   face->verticalRight - face->verticalLeft)+_string.size()-1;

double lineOrigin = 0;
if(f && f->getJustify().size()>1)
{
  string s = f->getJustify()[1];
  if(s == "BEGIN") lineOrigin = topToBottom ? -face->horizontalTop : 0;
```



```
  else if(s == "MIDDLE")
    lineOrigin =
      topToBottom ? totalSecondaryExtent/2 - face->horizontalTop :
      -totalSecondaryExtent/2;
  else if(s == "END")
    lineOrigin =
      topToBottom ? totalSecondaryExtent - face->horizontalTop :
      -totalSecondaryExtent;
}

glFrontFace(GL_CCW);
glDisable(GL_COLOR_MATERIAL);
glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, 1);
glDisable(GL_CULL_FACE);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glEnable(GL_BLEND);
glAlphaFunc(GL_GREATER, 0); // Pass if incoming alpha is greater that 0
glEnable(GL_ALPHA_TEST);
glNormal3d(0, 0, 1);

glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, face->textureId);

glBegin(GL_QUADS);
for(unsigned i=0; i<_string.size(); ++i)
{
  Vector2 pen(stringExtents[i] * primaryAlignment, lineOrigin);
  const double ajust = stringAjusts[i];

  for(unsigned j=0; j<stringGlyphs[i].size(); ++j)
  {
    const Glyph *g = stringGlyphs[i][j];

    if(!leftToRight)
      pen[0] -= g->horizontalAdvance*size+ajust;

    Vector2 layoutLowerLeft  = pen + g->horizontalLowerLeft * size;
    Vector2 layoutUpperRight = layoutLowerLeft + g->size * size;

    layoutLowerLeft[0]  *= compress;
    layoutUpperRight[0] *= compress;

    glTexCoord2d(g->textureUpperRight[0], g->textureUpperRight[1]);
    glVertex2d(layoutUpperRight[0], layoutUpperRight[1]);

    glTexCoord2d(g->textureLowerLeft[0], g->textureUpperRight[1]);
    glVertex2d(layoutLowerLeft[0], layoutUpperRight[1]);

    glTexCoord2d(g->textureLowerLeft[0], g->textureLowerLeft[1]);
    glVertex2d(layoutLowerLeft[0], layoutLowerLeft[1]);

    glTexCoord2d(g->textureUpperRight[0], g->textureLowerLeft[1]);
    glVertex2d(layoutUpperRight[0], layoutLowerLeft[1]);

    if(leftToRight)
      pen[0] += g->horizontalAdvance*size+ajust;
  }
```

```
        lineOrigin += lineAdvance;
      }
      glEnd();
    }

    void addFontPath(string p)
    {
      FontList::fontPaths.push_back(p);
    }
  }
}
```

Listing: **archon/x3d/server/texture.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_TEXTURE_H
#define ARCHON_X3D_TEXTURE_H

/*
 * \file texture.C
 *
 * X3D Texturing component
 *
 * NOTE: "X3DAppearanceChildNode" had to be moved from the Appearance
 * component to the Texture component to satisfy both inheritance and
 * containment hierachies.
 */

#include <archon/util/image.H>

#include <archon/x3d/server/network.H>
#include <archon/x3d/server/geoprop.H>

namespace Archon
{
  namespace X3D
  {
    struct TextureCoordinateNode: GeometricPropertyNode
    {
      static const NodeType *type;
```



```
    protected:
      TextureCoordinateNode() {}
    };


    struct TextureCoordinate: TextureCoordinateNode
    {
      static Ref<Node> instantiate(BackRef<ExecutionContext> c)
      {
        return new TextureCoordinate(c);
      }

      static const NodeType *type;
      const NodeType *getType() const { return type; }

      TextureCoordinate(BackRef<ExecutionContext> c): Node(c) {}

      const vector<Vector2> &getPoint() const { return point; }

    private:
      friend void initializeTextureComponent();

      vector<Vector2> point;

      EventSource pointChanged;

      Time pointStamp;
    };


    /**
     * "AppearanceChildNode" belongs to the X3D Appearance component
     * according to the specification but needs to be here to satisfy
     * both inheritance and containment hierachies. This is crucial when
     * initializing the runtime type information. It also allows the
     * component header files to be ordered according dependencies at
     * compile time.  Note: The proper order of header file inclusion
     * that satisfies both inheritance and containment hierachies is
     * stated in "x3d.H"/"x3d.H".
     */
    struct AppearanceChildNode: virtual Node
    {
      static const NodeType *type;

    protected:
      AppearanceChildNode() {}
    };


    struct TextureTransformNode: AppearanceChildNode
    {
      static const NodeType *type;

    protected:
      TextureTransformNode() {}
    };
```

```
struct TextureTransform: TextureTransformNode
{
  static Ref<Node> instantiate(BackRef<ExecutionContext> c)
  {
    return new TextureTransform(c);
  }

  static const NodeType *type;
  const NodeType *getType() const { return type; }

  TextureTransform(BackRef<ExecutionContext> c):
    Node(c),
    center(0, 0), rotation(0),
    scale(1, 1), translation(0, 0) {}

  const Vector2 &getCenter()          const { return center; }
  const double getRotation()          const { return rotation; }
  const Vector2 &getScale()           const { return scale; }
  const Vector2 &getTranslation()     const { return translation; }

private:
  friend void initializeTextureComponent();

  Vector2 center;
  double rotation;
  Vector2 scale;
  Vector2 translation;

  EventSource centerChanged;
  EventSource rotationChanged;
  EventSource scaleChanged;
  EventSource translationChanged;

  Time centerStamp;
  Time rotationStamp;
  Time scaleStamp;
  Time translationStamp;
};


struct TextureNode: AppearanceChildNode
{
  static const NodeType *type;

protected:
  TextureNode() {}
};


struct Texture2DNode: TextureNode
{
  static const NodeType *type;

  bool getRepeatS() const { return repeatS; }
  bool getRepeatT() const { return repeatT; }

protected:
```



```
  Texture2DNode(): repeatS(true), repeatT(true) {}

private:
  friend void initializeTextureComponent();

  bool repeatS;
  bool repeatT;
};


struct ImageTexture: Texture2DNode, UrlObject
{
  static Ref<Node> instantiate(BackRef<ExecutionContext> c)
  {
    return new ImageTexture(c);
  }

  static const NodeType *type;
  const NodeType *getType() const { return type; }

  ImageTexture(BackRef<ExecutionContext> c): Node(c) {}

private:
  friend void initializeTextureComponent();

  Loader::UriType getUriType() const
  {
    return Loader::uriType_Image;
  }
};


struct MovieTexture: Texture2DNode, UrlObject
{
  static Ref<Node> instantiate(BackRef<ExecutionContext> c)
  {
    return new MovieTexture(c);
  }

  static const NodeType *type;
  const NodeType *getType() const { return type; }

  MovieTexture(BackRef<ExecutionContext> c):
    Node(c), loop(false), speed(1), isActive(0) {}

private:
  friend void initializeTextureComponent();

  bool        loop;
  double      speed;
  Time        startTime;
  Time        stopTime;
  Time        duration;
  bool        isActive;

  EventSource loopChanged;
  EventSource speedChanged;
  EventSource startTimeChanged;
```

```
      EventSource stopTimeChanged;
      EventSource durationChanged;
      EventSource isActiveChanged;

      Time loopStamp;
      Time speedStamp;
      Time startTimeStamp;
      Time stopTimeStamp;

      Loader::UriType getUriType() const
      {
        return Loader::uriType_Movie;
      }
    };


    struct PixelTexture: Texture2DNode
    {
      static Ref<Node> instantiate(BackRef<ExecutionContext> c)
      {
        return new PixelTexture(c);
      }

      static const NodeType *type;
      const NodeType *getType() const { return type; }

      PixelTexture(BackRef<ExecutionContext> c): Node(c) {}

    private:
      friend void initializeTextureComponent();

      Image image;

      EventSource imageChanged;

      Time imageStamp;
    };
  }
}

#endif // ARCHON_X3D_TEXTURE_H
```

Listing: **archon/x3d/server/texture.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
```



```
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

/*
 * \file texture.C
 *
 * X3D Texturing component
 *
 * NOTE: "X3DAppearanceChildNode" had to be moved from the Appearance
 * component to the Texture component to satisfy both inheritance and
 * containment hierachies.
 */

#include <archon/x3d/server/field_type.H>
#include <archon/x3d/server/field.H>
#include <archon/x3d/server/texture.H>

namespace Archon
{
  namespace X3D
  {
    extern long nodeTypeIndexEnumerator;

    const NodeType *TextureCoordinateNode::type = 0;
    const NodeType *TextureCoordinate::type = 0;

    const NodeType *AppearanceChildNode::type = 0; // See above
    const NodeType *TextureTransformNode::type = 0;
    const NodeType *TextureTransform::type = 0;
    const NodeType *TextureNode::type = 0;
    const NodeType *Texture2DNode::type = 0;
    const NodeType *ImageTexture::type = 0;
    const NodeType *MovieTexture::type = 0;
    const NodeType *PixelTexture::type = 0;

    void initializeTextureComponent()
    {
      vector<const FieldBase *> fields;


      // TextureCoordinateNode

      TextureCoordinateNode::type =
        NodeType::newAbstract("TextureCoordinateNode", false, 0,
                              GeometricPropertyNode::type);


      // TextureCoordinate

      fields.push_back(newExposedField("point", MFVec2f::type,
                                       &TextureCoordinate::point,
                                       &TextureCoordinate::pointChanged,
                                       &TextureCoordinate::pointStamp));
      TextureCoordinate::type =
        NodeType::newConcrete("TextureCoordinate", TextureCoordinate::
             instantiate, &fields,
```

```
                        TextureCoordinateNode::type);
fields.clear();


// AppearanceChildNode

AppearanceChildNode::type =
  NodeType::newAbstract("AppearanceChildNode", false, 0,
                        Node::type);


// TextureTransformNode

TextureTransformNode::type =
  NodeType::newAbstract("TextureTransformNode", false, 0,
                        AppearanceChildNode::type);


// TextureTransform

fields.push_back(newExposedField("center", SFVec2f::type,
                                 &TextureTransform::center,
                                 &TextureTransform::centerChanged,
                                 &TextureTransform::centerStamp));
fields.push_back(newExposedField("rotation", SFFloat::type,
                                 &TextureTransform::rotation,
                                 &TextureTransform::rotationChanged,
                                 &TextureTransform::rotationStamp));
fields.push_back(newExposedField("scale", SFVec2f::type,
                                 &TextureTransform::scale,
                                 &TextureTransform::scaleChanged,
                                 &TextureTransform::scaleStamp));
fields.push_back(newExposedField("translation", SFVec2f::type,
                                 &TextureTransform::translation,
                                 &TextureTransform::translationChanged,
                                 &TextureTransform::translationStamp));
TextureTransform::type =
  NodeType::newConcrete("TextureTransform", TextureTransform::
       instantiate, &fields,
                        TextureTransformNode::type);
fields.clear();


// TextureNode

TextureNode::type =
  NodeType::newAbstract("TextureNode", false, 0,
                        AppearanceChildNode::type);
fields.clear();


// Texture2DNode

fields.push_back(newPrivateField("repeatS", SFBool::type,
                                 &Texture2DNode::repeatS));
fields.push_back(newPrivateField("repeatT", SFBool::type,
                                 &Texture2DNode::repeatT));
Texture2DNode::type =
```



```
  NodeType::newAbstract("Texture2DNode", false, &fields,
                        TextureNode::type);
fields.clear();


// ImageTexture

ImageTexture::type =
  NodeType::newConcrete("ImageTexture", ImageTexture::instantiate, &
       fields,
                        Texture2DNode::type, UrlObject::type);
fields.clear();


// MovieTexture

fields.push_back(newExposedField("loop", SFBool::type,
                                 &MovieTexture::loop,
                                 &MovieTexture::loopChanged,
                                 &MovieTexture::loopStamp));
fields.push_back(newExposedField("speed", SFFloat::type,
                                 &MovieTexture::speed,
                                 &MovieTexture::speedChanged,
                                 &MovieTexture::speedStamp));
fields.push_back(newExposedField("startTime", SFTime::type,
                                 &MovieTexture::startTime,
                                 &MovieTexture::startTimeChanged,
                                 &MovieTexture::startTimeStamp));
fields.push_back(newExposedField("stopTime", SFTime::type,
                                 &MovieTexture::stopTime,
                                 &MovieTexture::stopTimeChanged,
                                 &MovieTexture::stopTimeStamp));
fields.push_back(newEventOut("duration", SFTime::type,
                             &MovieTexture::duration,
                             &MovieTexture::durationChanged));
fields.push_back(newEventOut("isActive", SFBool::type,
                             &MovieTexture::isActive,
                             &MovieTexture::isActiveChanged));
MovieTexture::type =
  NodeType::newConcrete("MovieTexture", MovieTexture::instantiate, &
       fields,
                        Texture2DNode::type, UrlObject::type);
fields.clear();


// PixelTexture

fields.push_back(newExposedField("image", SFImage::type,
                                 &PixelTexture::image,
                                 &PixelTexture::imageChanged,
                                 &PixelTexture::imageStamp));
PixelTexture::type =
  NodeType::newConcrete("PixelTexture", PixelTexture::instantiate, &
       fields,
                        Texture2DNode::type);
fields.clear();
  }
}
```

```
}
```

Listing: **archon/x3d/server/time.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_TIME_H
#define ARCHON_X3D_TIME_H

/*
 * \file time.H
 *
 *  X3D Time component
 */

#include <archon/x3d/server/core.H>

namespace Archon
{
  namespace X3D
  {
    struct TimeDependentNode: virtual ChildNode
    {
      static const NodeType *type;

    protected:
      TimeDependentNode(): isPaused(false), loop(false), numLoops(1) {}

      friend void initializeTimeComponent();

      bool   isPaused;     // [out]
      Time   elapsedTime;  // [out]
      bool   loop;         // [in,out]   FALSE
      double numLoops;     // [in,out]   1     POSITIVE
      Time   startTime;    // [in,out]   0
      Time   stopTime;     // [in,out]   0
      Time   pauseTime;    // [in,out]   0
      Time   resumeTime;   // [in,out]   0

      EventSource isPausedChanged;
      EventSource elapsedTimeChanged;
```



```
      EventSource loopChanged;
      EventSource numLoopsChanged;
      EventSource startTimeChanged;
      EventSource stopTimeChanged;
      EventSource pauseTimeChanged;
      EventSource resumeTimeChanged;

      Time loopStamp;
      Time numLoopsStamp;
      Time startTimeStamp;
      Time stopTimeStamp;
      Time pauseTimeStamp;
      Time resumeTimeStamp;
    };

    struct TimeSensor: TimeDependentNode, SensorNode
    {
      static Ref<Node> instantiate(BackRef<ExecutionContext> c)
      {
        return new TimeSensor(c);
      }

      static const NodeType *type;
      const NodeType *getType() const { return type; }

      TimeSensor(BackRef<ExecutionContext>);
      ~TimeSensor();

      void tick();

    private:
      friend void initializeTimeComponent();

      Time   cycleInterval; // SFTime  [in,out]  1     POSITIVE
      Time   cycleTime;     // SFTime  [out]
      double fraction;      // SFFloat [out]
      Time   time;          // SFTime  [out]

      EventSource cycleIntervalChanged;
      EventSource cycleTimeChanged;
      EventSource fractionChanged;
      EventSource timeChanged;

      Time cycleIntervalStamp;

      void onRealized();
    };
  }
}

#endif // ARCHON_X3D_TIME_H
```

Listing: **archon/x3d/server/time.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
```

```
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

/*
 * \file time.C
 *
 *  X3D Time component
 */

#include <math.h>

#include <iostream>

#include <archon/x3d/server/field_type.H>
#include <archon/x3d/server/field.H>
#include <archon/x3d/server/time.H>
#include <archon/x3d/server/scene.H>
#include <archon/x3d/server/server.H>

namespace Archon
{
  namespace X3D
  {
    extern long nodeTypeIndexEnumerator;

    const NodeType *TimeDependentNode::type = 0;
    const NodeType *TimeSensor::type = 0;

    void initializeTimeComponent()
    {
      vector<const FieldBase *> fields;

      // TimeDependentNode

      fields.push_back(newEventOut("isPaused", SFBool::type,
                                   &TimeDependentNode::isPaused,
                                   &TimeDependentNode::isPausedChanged));
      fields.push_back(newEventOut("elapsedTime", SFTime::type,
                                   &TimeDependentNode::elapsedTime,
                                   &TimeDependentNode::elapsedTimeChanged));
      fields.push_back(newExposedField("loop", SFBool::type,
                                       &TimeDependentNode::loop,
                                       &TimeDependentNode::loopChanged,
                                       &TimeDependentNode::loopStamp));
      fields.push_back(newExposedField("numLoops", SFFloat::type,
                                       &TimeDependentNode::numLoops,
```



```
                                       &TimeDependentNode::numLoopsChanged,
                                       &TimeDependentNode::numLoopsStamp));
      fields.push_back(newExposedField("startTime", SFTime::type,
                                       &TimeDependentNode::startTime,
                                       &TimeDependentNode::startTimeChanged,
                                       &TimeDependentNode::startTimeStamp));
      fields.push_back(newExposedField("stopTime", SFTime::type,
                                       &TimeDependentNode::stopTime,
                                       &TimeDependentNode::stopTimeChanged,
                                       &TimeDependentNode::stopTimeStamp));
      fields.push_back(newExposedField("pauseTime", SFTime::type,
                                       &TimeDependentNode::pauseTime,
                                       &TimeDependentNode::pauseTimeChanged,
                                       &TimeDependentNode::pauseTimeStamp));
      fields.push_back(newExposedField("resumeTime", SFTime::type,
                                       &TimeDependentNode::resumeTime,
                                       &TimeDependentNode::resumeTimeChanged,
                                       &TimeDependentNode::resumeTimeStamp));
      TimeDependentNode::type =
        NodeType::newAbstract("TimeDependentNode", false,
                              &fields, ChildNode::type);
      fields.clear();


      // TimeSensor

      fields.push_back(newExposedField("cycleInterval", SFTime::type,
                                       &TimeSensor::cycleInterval,
                                       &TimeSensor::cycleIntervalChanged,
                                       &TimeSensor::cycleIntervalStamp));
      fields.push_back(newEventOut("cycleTime", SFTime::type,
                                   &TimeSensor::cycleTime,
                                   &TimeSensor::cycleTimeChanged));
      fields.push_back(newEventOut("fraction", SFFloat::type,
                                   &TimeSensor::fraction,
                                   &TimeSensor::fractionChanged));
      fields.push_back(newEventOut("time", SFTime::type,
                                   &TimeSensor::time,
                                   &TimeSensor::timeChanged));
      TimeSensor::type =
        NodeType::newConcrete("TimeSensor", TimeSensor::instantiate,
                              &fields, TimeDependentNode::type, SensorNode::
                                  type);
      fields.clear();
    }

    TimeSensor::TimeSensor(BackRef<ExecutionContext> c):
      Node(c), cycleInterval(1.0), fraction(0)
    {
    }

    TimeSensor::~TimeSensor()
    {
      if(!isRealized()) return;
      try{ context->removeTimeSensor(this); }
      catch(ForwardDestroyedException &) {}
    }
```

```
void TimeSensor::onRealized()
{
  context->addTimeSensor(this);
}

/**
 * \todo TimeSensors that are initially enabled at load time don't
 * start properly. That is, they don't start from the beginning of
 * a cycle. I guess they should. Maybe the loader should
 * initialize them with startTime=now or maybe the case
 * "startTime=0" should be handles in a special way.
 *
 * \todo Implement the rest of the TimeSensor functionality.
 */
void TimeSensor::tick()
{
  if(!getEnabled()) return;
  Time now = context->server->getNextTimeStamp();
  if(!isActive)
  {
    if(now<startTime ||
       now>=stopTime && stopTime > startTime ||
       now>=startTime+cycleInterval && !loop) return;

    // Activate
    isActive = true;

    // Skip an integer number of cycleInterval's so that we arrive
    // at the greatest cycleTime less than now. This seems to give
    // undesired effects for TimeSensors with startTime=0 (think
    // it over agian)
    long double phase;
    long double period;
    Time t=now-startTime;
    t.get(phase);
    cycleInterval.get(period);
    cycleTime = startTime + Time(floor(double(phase/period)) * double(
         period));

    {
      Event event(new SimpleValue<bool>(SFBool::type, isActive), now);
      isActiveChanged.cascadeEvent(&event);
    }
    {
      Event event(new SimpleValue<Time>(SFTime::type, cycleTime), now);
           /// \todo We should increase the time
      cycleTimeChanged.cascadeEvent(&event);
    }

    //cerr << "Activated '" << getName() << "'\n";
  }
  else if(now>=stopTime && stopTime > startTime ||
          now>=cycleTime+cycleInterval && !loop)
  {
    // Deactivate
    time = loop ? stopTime : min(stopTime, cycleTime+cycleInterval);
    fraction = 1;
    isActive = false;
```



```
    {
      Event event(new SimpleValue<Time>(SFTime::type, time), now); /// \
           todo We should increase the time
      timeChanged.cascadeEvent(&event);
    }
    {
      Event event(new SimpleValue<double>(SFFloat::type, fraction), now
           ); /// \todo We should increase the time
      fractionChanged.cascadeEvent(&event);
    }
    {
      Event event(new SimpleValue<bool>(SFBool::type, isActive), now);
      isActiveChanged.cascadeEvent(&event);
    }
    //cerr << "De-activated '" << getName() << "'\n";
    return;
  }

  if(now>=cycleTime+cycleInterval)
  {
    // Next loop
    cycleTime += cycleInterval;

    Event event(new SimpleValue<Time>(SFTime::type, cycleTime), now);
         /// \todo We should increase the time
    cycleTimeChanged.cascadeEvent(&event);
  }

  long double phase;
  long double period;
  Time t = now-cycleTime;
  t.get(phase);
  cycleInterval.get(period);

  time = now;
  fraction = phase/period;

  {
    Event event(new SimpleValue<Time>(SFTime::type, time), now); /// \
         todo We should increase the time
    timeChanged.cascadeEvent(&event);
  }
  {
    Event event(new SimpleValue<double>(SFFloat::type, fraction), now);
         /// \todo We should increase the time
    fractionChanged.cascadeEvent(&event);
  }
   }
  }
}
```

Listing: **archon/x3d/server/type.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
```

```
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_TYPE_H
#define ARCHON_X3D_TYPE_H

/**
 * \file type.H
 *
 * X3D runtime type information for nodes and fields
 */

#include <string>
#include <map>
#include <list>

#include <archon/util/logger.H>
#include <archon/util/ref.H>
#include <archon/util/time.H>

#include <archon/x3d/server/exception.H>

namespace Archon
{
  namespace X3D
  {
    using namespace std;
    using namespace Utilities;

    struct ValueBase;
    struct Event;
    struct Route;
    struct RouteTail;
    struct Node;
    struct NodeType;

    struct ExecutionContext; // Defined in scene.H

    /**
     * Runtime representation of an X3D field type.
     *
     * Examples of field types are: SFBool, MFVec2f...
     */
    struct FieldType
    {
      FieldType(unsigned index, string name, const FieldType *singleType=0);
```



```
      unsigned getIndex() const { return index; }
      string getName() const { return name; }
      bool getIsMultiple() const { return singleType; }
      const FieldType *getSingleType() const { return singleType; }

      /**
       * Fetch a predefined or user defined node type.
       *
       * Only concrete node types can be found by this method. This is
       * appropriate for the X3D parsers.
       *
       * \param name Is the UTF8 encoded field type name such as
       * SFBool, MFVec2f etc.
       */
      static const FieldType *lookup(string name);

    private:
      unsigned index;
      string name;
      const FieldType *singleType;
    };

    /**
     * Virtual base class for all field definitions. This includes the
     * field wrappers that refer to real class level variables as well
     * as the custom fields that can be contained in Script node
     * instanses (the user specified ones).
     *
     * \sa CustomFieldBase
     */
    struct FieldBase
    {
      unsigned long getId() const { return id; }
      string getName() const { return name; }
      const FieldType *getType() const { return type; }
      bool getIsEventTarget() const { return isEventTarget; }
      bool getIsEventSource() const { return isEventSource; }

      /**
       * Fetch the X3D field associated with the specified field ID.
       *
       * \sa getId
       *
       * \param n Must be specified if you want to be able to retrieve
       * custum fields.
       */
      static const FieldBase *fetch(unsigned long fieldId, const Node *n=0);

      /**
       * Flag that specifies that the 'set_*' and '*_changed' variants
       * of this field should not be defined, only the plain name as
       * specified.
       *
       * \sa getFlags
       */
      static const unsigned NOVARIANT = 1;
```

```
/**
 * Returns the flags set for this field.
 *
 * \sa NOVARIANT
 */
unsigned getFlags() const { return flags; }

virtual Ref<ValueBase> get(const Node *) const = 0;

/**
 * IF cascade IS FALSE:
 *
 * Assign the specified event value to this field, but do not
 * cascade the event on through the associated routes.
 *
 * Must be called with a lock on the node and the node must not
 * be realized.
 *
 * IF cascade IS TRUE:
 *
 * Assign the specified event value to this field, and cascade
 * the event on through all associated routes.
 *
 * Must be called with a scenegraph write lock, and never with a
 * lock on any node. Also, this method may never be called for an
 * un-realized node.
 */
virtual void handleEvent(Node *, const Event *, bool cascade) const
      = 0;

/**
 * Is this field or any field of child nodes of this field changed?
 * Child nodes of course are only relevant for node fields.
 *
 * Must be called with a scenegraph read lock.
 */
virtual bool changedSince(const Node *n, const Time &) const = 0;

/**
 * Add a route to this field through EventSource::addRoute.
 *
 * \sa EventSource::addRoute
 *
 * \note This method should only be called from descendants of
 * RouteTail.
 */
virtual bool addRoute(Node *, const RouteTail *) const = 0;

/**
 * Remove a route from this field through EventSource::delRoute.
 *
 * \sa EventSource::delRoute
 *
 * \note This method should only be called from descendants of
 * RouteTail.
 */
virtual void delRoute(Node *, const RouteTail *) const = 0;
```



```
  /**
   * Remove a matching route from this field through
   * EventSource::delRouteMAtch.
   *
   * \sa EventSource::delRouteMatch
   *
   * \note This method should only be called from descendants of
   * RouteTail.
   */
  virtual bool delRouteMatch(Node *, const RouteTail *) const = 0;

  /**
   * Must be called with a scenegraph read lock.
   */
  virtual string dump(const Node *n, int level) const = 0;

  virtual ~FieldBase() {}

protected:
  /**
   * \param flasg Is a bitwise or of zero or more of the flags:
   * NOVARIANT.
   *
   * \sa NOVARIANT
   */
  FieldBase(string name, const FieldType *,
           bool isEventTarget, bool isEventSource, unsigned flags);

  /**
   * Since this is a virtual base class, abstract subclasses will
   * never call a constructor in this class. Still, if we only
   * provided the a constructor taking arguments then the abstract
   * subclasses would be forced to call it and invent dummy values
   * for the arguments, and completely in vain since the code will
   * never be executed, Thast why we provide this one. DONT EVER
   * USE IT!
   */
  FieldBase() {}

  bool isEventTarget;
  bool isEventSource;

  friend struct Node;
  friend struct NodeType;

  virtual void forwardClear(Node *) const = 0;

  unsigned long id;
  string name;
  const FieldType *type;

  /**
   * \sa NOVARIANT
   */
  unsigned flags;
};
```

```
/**
 * Base class for all kinds of fields which are dynamic length
 * sequences. That is, any field type whos name starts with MF.
 */
struct SequenceFieldBase: virtual FieldBase
{
  virtual unsigned getSize(const Node *) const = 0;
  virtual Ref<ValueBase> getAt(const Node *, unsigned index) const = 0;
  virtual void setAt(Node *, const ValueBase *, unsigned index, Time,
       bool cascade) const = 0;
  virtual void add(Node *, const ValueBase *, Time, bool cascade) const
       = 0;
  virtual void remove(Node *, const ValueBase *, Time, bool cascade)
       const = 0;
};


/**
 * Base class for all kinds of fields of type SFNode or MFNode.
 */
struct NodeFieldBase: virtual FieldBase
{
  const NodeType *getNodeType() const { return nodeType; }

protected:
  NodeFieldBase(const NodeType *nodeType): nodeType(nodeType)
  {
    if(!nodeType)
      ARCHON_THROW1(InternalException,
                    "Refering to uninitialized node type");
  }

private:
  friend struct Node;

  /**
   * Returns true if the target field "has room" for the new node.
   * If the target field is multi-valued (MFNode) it always has
   * room for another node, but if the target field is
   * single-valued (SFNode) and a node has previously been
   * injected, then there is no room and false will be returned.
   *
   * Must be called with a lock on the specified node, and never
   * for a realized node.
   */
  virtual bool inject(Node *, Node *childNode, Time, bool cascade) const
       = 0;

  const NodeType *nodeType;
};

/**
 * Base class for all kinds of fields of type MFNode.
 */
struct NodeSequenceFieldBase: NodeFieldBase, SequenceFieldBase
{
protected:
  NodeSequenceFieldBase(const NodeType *nodeType):
```



```
    NodeFieldBase(nodeType) {}
};

/**
 * Runtime representation of an X3D node type.
 *
 * Examples of field types are: Group, Transform...
 *
 * Node types may inherit fields from other node type through
 * multiple iheritance. There are two restrictions though: The
 * inheritance graph may not by cyclic (of cause), and the names
 * of all local and inherited fields must be individually unique
 * in every node type. This includes all set_* and *_changed
 * variants.
 *
 * There is a last restriction relating to the way XML encoded X3D
 * documents are parsed: Among all local and inherited fields
 * which are private or exposed and of node type (SFNode and
 * MFNode), no two may have the same node type associated with
 * them. Neither may the node type of one node field be derived in
 * one or more steps from the node type of another such node
 * field.  This enables the XML parser to uniquely determine into
 * which field a sub node must be placed.
 */
struct NodeType
{
  static const int maxIndexBits = 7;

  unsigned long getIndex() const { return index; }

  /**
   * \return the total number of X3D node types.
   */
  static unsigned long getTotalNumber();

  /**
   * Fetch an X3D node type by index.
   */
  static const NodeType *get(unsigned long index);

  /**
   * Lookup an X3D node type by name.
   *
   * \param nodeTypeName Is UTF8 encoded.
   */
  static const NodeType *lookup(const string &nodeTypeName);

  Ref<Node> instantiate(BackRef<ExecutionContext> context) const;

  string getName() const { return name; }

  bool isAbstract() const { return !instanceMaker; }
  bool isVirtualBase() const { return virtualBase; }
  bool isInternalUseOnly() const { return internalUseOnly; }
  bool isDerivedFrom(const NodeType *baseType) const;

  /**
   * May this node be instantiated through the X3D parser and the
```

```
   * SAI?
   */
  bool isUserInstantiable() const
  {
    return instanceMaker && !internalUseOnly;
  }

  unsigned long getNumberOfImmediateFields() const
  {
    return fields.size();
  }

  const FieldBase *getImmediateField(unsigned long fieldIndex) const
  {
    if(fieldIndex >= fields.size())
      ARCHON_THROW1(InternalException, "Index out of bound");
    return fields[fieldIndex];
  }

  /**
   * Like NodeType::lookupField except that this method does not
   * include the fields of the base node types from which this one
   * is derived. Also, this method does not detect the 'set_*' and
   * '*_changed' variants of the field names, only the base names
   * are included in the search.
   */
  const FieldBase *lookupImmediateField(string name) const;

  /**
   * Search for a field of the specified name in this node type.
   * 'set_*' and '*_changed' variants are tryed automatically.
   *
   * \return 0 if not found.
   *
   * \note This method obviously cannot find custom fields. If you
   * need to search among both static and custom fields then use
   * Node::lookupField instead.
   */
  const FieldBase *lookupField(string name) const;

  /**
   * Seach for a private or exposed node fields whos type is a
   * base type of the argument type. There should at all times be
   * at most one such field in a node (the uniqueness is checked
   * by the constructor).
   *
   * \return 0 if not found. If found the returned type will
   * either be a descendant of NodeFieldBase or a descendant of
   * NodeSequenceFieldBase.
   */
  const FieldBase *lookupNodeField(const NodeType *) const;


  static const NodeType *newAbstract(string name, bool virtualBase,
                                     const vector<const FieldBase *> *
                                         fields);
  static const NodeType *newAbstract(string name, bool virtualBase,
```



```
                                     const vector<const FieldBase *> *
                                         fields,
                                     const NodeType *derivedFrom1);
  static const NodeType *newAbstract(string name, bool virtualBase,
                                     const vector<const FieldBase *> *
                                         fields,
                                     const NodeType *derivedFrom1,
                                     const NodeType *derivedFrom2);
  static const NodeType *newConcrete(string name,
                                     Ref<Node> (*instanceMaker)(BackRef<
                                         ExecutionContext>),
                                     const vector<const FieldBase *> *
                                         fields,
                                     const NodeType *derivedFrom1);
  static const NodeType *newConcrete(string name,
                                     Ref<Node> (*instanceMaker)(BackRef<
                                         ExecutionContext>),
                                     const vector<const FieldBase *> *
                                         fields,
                                     const NodeType *derivedFrom1,
                                     const NodeType *derivedFrom2);

  /**
   * Must be called with a scenegraph read lock.
   */
  bool changedSince(const Node *, const Time &) const;

  void describe(Logger *) const;
  static void describeAll(Logger *);

  /**
   * Must be called with a scenegraph read lock.
   */
  string dumpFields(const Node *, int level) const;

  /**
   * Must be called with a scenegraph read lock.
   */
  string dumpNode(const Node *, int level) const;

  typedef vector<const NodeType *>::const_iterator iterator;
  iterator derivedFromBegin() const;
  iterator derivedFromEnd() const;

private:
  unsigned long index;
  const string name;
  bool virtualBase;
  bool internalUseOnly;
  vector<const NodeType *> derivedFrom;
  vector<const FieldBase *> fields;
  Ref<Node> (*instanceMaker)(BackRef<ExecutionContext>);
  map<string, const FieldBase *> fieldMap; // Fast access to all local
       and inherited fields

  void addFields(vector<pair<const FieldBase *, const NodeType *> > &)
       const;
  void initFieldMap(map<string, const FieldBase *> &,
```

```
                    const string &nodeType) const;

      /**
       * Create a new node type which will be abstract if
       * "instanceMaker" is null. Making a node type abstract has the
       * side effect that X3D parsers will will not recognize is as
       * avalid node if an X3D file.
       *
       * This fact may be used to define node types which to the
       * compiler are not abstract but according to the runtime type
       * information are. This effectivle allows you to define
       * concrete and instantiable node types which the parsers should
       * not know about.
       */
      NodeType(string name, bool virtualBase,
               const vector<const NodeType *> *derivedFrom,
               const vector<const FieldBase *> *fields,
               Ref<Node> (*instanceMaker)(BackRef<ExecutionContext>));


      friend struct Node;

      void forwardClear(Node *) const;
    };
  }
}

#endif // ARCHON_X3D_TYPE_H
```

Listing: **archon/x3d/server/type.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

/**
 * \file type.C
 *
 * X3D runtime type information for nodes and fields
 */

#include <archon/util/term.H>
#include <archon/util/text.H>
```



```
#include <archon/x3d/server/type.H>
#include <archon/x3d/server/node.H>
#include <archon/x3d/server/scene.H>
#include <archon/x3d/server/custom_field.H>

namespace Archon
{
  using namespace Utilities;

  namespace X3D
  {
    namespace
    {
      /**
       * \todo Get rid of these non-fundamental global vars.
       */
      vector<const NodeType *> nodeTypes;

      map<string, const NodeType *>  nodeTypeMap;
      map<string, const FieldType *> fieldTypeMap;
    }

    void NodeType::addFields(vector<pair<const FieldBase *,
                             const NodeType *> > &l) const
    {
      for(unsigned i=0; i<derivedFrom.size(); ++i)
        derivedFrom[i]->addFields(l);
      for(unsigned i=0; i<fields.size(); ++i)
        l.push_back(make_pair(fields[i], this));
    }

    void NodeType::initFieldMap(map<string, const FieldBase *> &m,
                                const string &nodeType) const
    {
      for(unsigned i=0; i<derivedFrom.size(); ++i)
      {
        derivedFrom[i]->initFieldMap(m, nodeType);
      }
      for(unsigned i=0; i<fields.size(); ++i)
      {
        const FieldBase *f = fields[i];
        if(!m.insert(make_pair(f->getName(), f)).second ||
           !(f->getFlags()&FieldBase::NOVARIANT) &&
           (f->getIsEventTarget() &&
            !m.insert(make_pair("set_" + f->getName(), f)).second ||
            f->getIsEventSource() &&
            !m.insert(make_pair(f->getName() + "_changed", f)).second))
          ARCHON_THROW1(InternalException,
                        "Ambiguous field '" + f->getName() +
                        "' for node type '" + nodeType + "'");
      }
    }

    NodeType::NodeType(string name,
                       bool virtualBase,
                       const vector<const NodeType *> *d,
                       const vector<const FieldBase *> *f,
```

```
                  Ref<Node> (*instanceMaker)(BackRef<ExecutionContext
                       >)):
  index(nodeTypes.size()), name(name), virtualBase(virtualBase),
  internalUseOnly(false), instanceMaker(instanceMaker)
{
  if(nodeTypes.size() == (1 << maxIndexBits) - 1)
      ARCHON_THROW1(InternalException, "Too many node types");

  if(d) derivedFrom = *d;
  if(f)
  {
    fields = *f;
    for(unsigned i=0; i<fields.size(); ++i)
    {
      unsigned long j = i;
      j <<= maxIndexBits;
      j |= index;
      const_cast<FieldBase *>(fields[i])->id = j;
    }
  }

  // Verify uniqueness of all local and inherited field names
  initFieldMap(fieldMap, name);

  /*
   * Verify type uniqueness for private and exposed node fields.
   * That is, no two private and/or exposed node fields may have types
   * such that one is derived from the other. This is necessary due
   * to the fact that the XML encodeing does not explicitly name the
   * target fields for child nodes.
   */
  for(map<string, const FieldBase *>::iterator i = fieldMap.begin();
      i != fieldMap.end(); ++i)
    for(map<string, const FieldBase *>::iterator j = fieldMap.begin();
        j != fieldMap.end(); ++j)
      if(i->second != j->second)
        if(const NodeFieldBase *f1 =
           dynamic_cast<const NodeFieldBase *>(i->second))
          if(const NodeFieldBase *f2 =
             dynamic_cast<const NodeFieldBase *>(j->second))
            if(f1->getIsEventTarget() == f1->getIsEventSource() &&
               f2->getIsEventTarget() == f2->getIsEventSource() &&
               f1->getNodeType()->isDerivedFrom(f2->getNodeType()))
              ARCHON_THROW1(InternalException,
                            "Type clash between node fields '" +
                            f1->getName() + "' and '" +
                            f2->getName() + "'");

  if(!nodeTypeMap.insert(make_pair(name, this)).second)
    ARCHON_THROW1(InternalException,
                  "Redefinition of node type '" + name + "'");

  nodeTypes.push_back(this);
}

unsigned long NodeType::getTotalNumber()
{
  return nodeTypes.size();
```



```
}

const NodeType *NodeType::get(unsigned long i)
{
  if(i >= nodeTypes.size())
    ARCHON_THROW1(ArgumentException, "Index out of bound");
  return nodeTypes[i];
}

const NodeType *NodeType::lookup(const string &n)
{
  map<string, const NodeType *>::iterator i = nodeTypeMap.find(n);
  return i==nodeTypeMap.end() ? 0 : i->second;
}

Ref<Node> NodeType::instantiate(BackRef<ExecutionContext> c) const
{
  if(!instanceMaker)
    ARCHON_THROW1(ArgumentException,
                  "Attempt to instantiate abstract node type");
  return (*instanceMaker)(c);
}

bool NodeType::isDerivedFrom(const NodeType *baseType) const
{
  if(this == baseType) return true;
  for(unsigned i=0; i<derivedFrom.size(); ++i)
    if(derivedFrom[i]->isDerivedFrom(baseType)) return true;
  return false;
}


const FieldBase *FieldBase::fetch(unsigned long i, const Node *n)
{
  unsigned long j = i & (1 << NodeType::maxIndexBits) - 1;
  if(j >= nodeTypes.size()) return 0;
  const NodeType *t = nodeTypes[j];
  unsigned long k = i >> NodeType::maxIndexBits;
  if(k < t->getNumberOfImmediateFields()) return t->getImmediateField(k);
  if(!n || j != n->getType()->getIndex()) return 0;
  const CustomFieldNode *m = dynamic_cast<const CustomFieldNode *>(n);
  if(!m) return 0;
  return m->getCustomField(k - t->getNumberOfImmediateFields()).get();
}


FieldBase::FieldBase(string name, const FieldType *type,
                     bool isEventTarget, bool isEventSource, unsigned
                          flags):
  isEventTarget(isEventTarget), isEventSource(isEventSource),
  id(0), name(name), type(type), flags(flags)
{
}

const FieldBase *NodeType::lookupField(string name) const
{
  map<string, const FieldBase *>::const_iterator i =
    fieldMap.find(name);
```

```
  return i==fieldMap.end() ? 0 : i->second;
}

const FieldBase *NodeType::lookupImmediateField(string name) const
{
  for(unsigned i=0; i<fields.size(); ++i)
  {
    const FieldBase *f = fields[i];
    if(f->getName() == name) return f;
  }
  return 0;
}

const FieldBase *NodeType::lookupNodeField(const NodeType *nodeType)
     const
{
  for(unsigned i=0; i<fields.size(); ++i)
  {
    if(const NodeFieldBase *f =
       dynamic_cast<const NodeFieldBase *>(fields[i]))
    {
      if(f->getIsEventTarget() == f->getIsEventSource() &&
         nodeType->isDerivedFrom(f->getNodeType())) return f;
    }
  }
  for(unsigned i=0; i<derivedFrom.size(); ++i)
  {
    const FieldBase *f = derivedFrom[i]->lookupNodeField(nodeType);
    if(f) return f;
  }
  return 0;
}

const NodeType *NodeType::
newAbstract(string name, bool virtualBase,
            const vector<const FieldBase *> *fields)
{
  return new NodeType(name, virtualBase, 0, fields, 0);
}

const NodeType *NodeType::
newAbstract(string name, bool virtualBase,
            const vector<const FieldBase *> *fields,
            const NodeType *derivedFrom1)
{
  if(!derivedFrom1)
    ARCHON_THROW1(InternalException,
                  "Deriving from uninitialized node type");
  vector<const NodeType *> derivedFrom;
  derivedFrom.push_back(derivedFrom1);
  return new NodeType(name, virtualBase, &derivedFrom, fields, 0);
}

const NodeType *NodeType::
newAbstract(string name, bool virtualBase,
            const vector<const FieldBase *> *fields,
            const NodeType *derivedFrom1,
            const NodeType *derivedFrom2)
```



```
{
  if(!derivedFrom1 || !derivedFrom2)
    ARCHON_THROW1(InternalException,
                  "Deriving from uninitialized node type");
  vector<const NodeType *> derivedFrom;
  derivedFrom.push_back(derivedFrom1);
  derivedFrom.push_back(derivedFrom2);
  return new NodeType(name, virtualBase, &derivedFrom, fields, 0);
}

const NodeType *NodeType::
newConcrete(string name,
            Ref<Node> (*instanceMaker)(BackRef<ExecutionContext>),
            const vector<const FieldBase *> *fields,
            const NodeType *derivedFrom1)
{
  if(!derivedFrom1)
    ARCHON_THROW1(InternalException,
                  "Deriving from uninitialized node type");
  vector<const NodeType *> derivedFrom;
  derivedFrom.push_back(derivedFrom1);
  return new NodeType(name, false, &derivedFrom, fields, instanceMaker);
}

const NodeType *NodeType::
newConcrete(string name,
            Ref<Node> (*instanceMaker)(BackRef<ExecutionContext>),
            const vector<const FieldBase *> *fields,
            const NodeType *derivedFrom1,
            const NodeType *derivedFrom2)
{
  if(!derivedFrom1 || !derivedFrom2)
    ARCHON_THROW1(InternalException,
                  "Deriving '" + name + "' from uninitialized node type"
                       );
  vector<const NodeType *> derivedFrom;
  derivedFrom.push_back(derivedFrom1);
  derivedFrom.push_back(derivedFrom2);
  return new NodeType(name, false, &derivedFrom, fields, instanceMaker);
}

void NodeType::forwardClear(Node *n) const
{
  for(unsigned i=0; i<fields.size(); ++i)
    fields[i]->forwardClear(n);
  for(unsigned i=0; i<derivedFrom.size(); ++i)
    derivedFrom[i]->forwardClear(n);
}

bool NodeType::changedSince(const Node *n, const Time &t) const
{
  for(unsigned i=0; i<fields.size(); ++i)
    if(fields[i]->changedSince(n, t)) return true;
  for(unsigned i=0; i<derivedFrom.size(); ++i)
    if(derivedFrom[i]->changedSince(n, t)) return true;
  return false;
}
```

```
string NodeType::dumpFields(const Node *n, int level) const
{
  string s;
  for(unsigned i=0; i<derivedFrom.size(); ++i)
    s += derivedFrom[i]->dumpFields(n, level);
  for(unsigned i=0; i<fields.size(); ++i)
    s += fields[i]->dump(n, level);
  return s;
}

string NodeType::dumpNode(const Node *n, int level) const
{
  string indent(2*level, ' ');
  string s = indent;
  if(!n->getName().empty()) s += "DEF \"" + n->getName() + "\" ";
  s += n->getType()->getName() + " // [" +
    Text::toString(n->getUseCount()) + "]\n";
  s += indent + "{\n";
  s += dumpFields(n, level+1);
  s += indent + "}\n";
  return s;
}

void NodeType::describe(Logger *logger) const
{
  string header = getName();
  if(derivedFrom.size())
  {
    header += ": ";
    for(unsigned i=0; i<derivedFrom.size(); ++i)
    {
      if(i) header += ", ";
      header += derivedFrom[i]->getName();
    }
  }
  if(!instanceMaker) header += " // abstract";
  vector<pair<const FieldBase *, const NodeType *> > l;
  addFields(l);
  vector<double> columnWidthFractions;
  columnWidthFractions.push_back(0.1);
  columnWidthFractions.push_back(0.1);
  columnWidthFractions.push_back(0.2);
  columnWidthFractions.push_back(0.2);
  columnWidthFractions.push_back(0.2);
  columnWidthFractions.push_back(0.2);
  Text::Table<string> table(l.size(), columnWidthFractions);
  for(unsigned i=0; i<l.size(); ++i)
  {
    const FieldBase *f = l[i].first;
    table(i, 0) = "  " + f->getType()->getName();
    table(i, 1) = "[" +
      string(f->getIsEventTarget()?"in":"") +
      (f->getIsEventTarget() && f->getIsEventSource()?",":"") +
      (f->getIsEventSource()?"out":"") + "]";
    table(i, 2) = f->getName();
    table(i, 3) = "<default value>";
    if(const NodeFieldBase *nf =
       dynamic_cast<const NodeFieldBase *>(f))
```



```
      table(i, 4) = nf->getNodeType()->getName();
    else
      table(i, 4) = "<domain>";
    if(l[i].second != this) table(i, 5) = "// " + l[i].second->getName();
  }
  logger->log(header + "\n{\n" +
              Text::format(table, Term::getWidth(), 2, false) + "}\n");
}

void NodeType::describeAll(Logger *l)
{
  for(unsigned i=0; i<nodeTypes.size(); ++i) nodeTypes[i]->describe(l);
}

NodeType::iterator NodeType::derivedFromBegin() const
{
  return derivedFrom.begin();
}

NodeType::iterator NodeType::derivedFromEnd() const
{
  return derivedFrom.end();
}


FieldType::FieldType(unsigned index, string name,
                     const FieldType *singleType):
  index(index), name(name), singleType(singleType)
{
  if(!fieldTypeMap.insert(make_pair(name, this)).second)
    ARCHON_THROW1(InternalException,
                  "Redefinition of field type '" +
                  name + "'");
}

const FieldType *FieldType::lookup(string name)
{
  map<string, const FieldType *>::iterator i = fieldTypeMap.find(name);
  return i==fieldTypeMap.end() ? 0 : i->second;
}
  }
}
```

Listing: **archon/x3d/server/view.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
```

```
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_VIEW_H
#define ARCHON_X3D_VIEW_H

#include <stack>

#include <archon/math/vector.H>
#include <archon/math/geometry.H>

#include <archon/x3d/server/exception.H>
#include <archon/x3d/server/x3d.H>
#include <archon/x3d/server/scene.H>

namespace Archon
{
  namespace X3D
  {
    struct Viewer
    {
      struct InternalException: X3D::InternalException
      {
        InternalException(string l, string m): Utilities::Exception(l, m) {}
      };

      Viewer(Ref<Server>,
             int resolutionX, int resolutionY,
             bool useMipmapedTextures,
             string initialViewpointName,
             const CoordSystem3 &defaultViewpoint,
             double fieldOfView,
             double depthOfRotation,
             const Vector4 &backgroundColor,
             int subdivisionX, int subdivisionY,
             bool showNormals,
             bool wireframeMode, bool enableTexture);

      /**
       * Attempt to bind the viewpoint with the specified name.
       *
       * \return True if a viewpoint exists with that name.
       */
      bool bindViewpoint(string name);

      /**
       * Reset viewpoint back to the way it was initially. This
       * requires that you pass the same 'initialViewpointName' and
       * 'defaultViewpoint' as you did to the constructor.
       */
      void resetViewpoint(string initialViewpointName,
                          const CoordSystem3 &defaultViewpoint);

      bool view(bool headLight = true, bool showLightSources = false,
```



```
                bool sensorModeActive = false, bool pointingDeviceActive =
                     false,
                const Vector2 *newMousePosition = 0);

      bool getWireframeMode() { return wireframeMode; }
      void setWireframeMode(bool m = true) { wireframeMode = m; }

      bool getEnableTexture() { return enableTexture; }
      void setEnableTexture(bool m = true) { enableTexture = m; }

      bool getShowNormals() { return renderConfig.showNormals; }
      void setShowNormals(bool m = true)
      {
        renderConfig.showNormals = m;
        forcedShapeCacheUpdate = true;
      }

      bool getSeparateSpecularColorMode() { return separateSpecularColorMode
           ; }
      void setSeparateSpecularColorMode(bool m = true) {
           separateSpecularColorMode = m; }

      bool getLightModelLocalViewerMode() { return lightModelLocalViewerMode
           ; }
      void setLightModelLocalViewerMode(bool m = true) {
           lightModelLocalViewerMode = m ; }

      CoordSystem3 &manipViewCoordSystem()
      {
        boundViewpoint = 0;
        return viewCoordSystem;
      }

      double getDepthOfRotation()
      {
        return depthOfRotation;
      }

      void setDepthOfRotation(double d)
      {
        boundViewpoint = 0;
        depthOfRotation = d;
        renderConfig.normalLength = d/10;
      }

    private:
      unsigned maxLights;

      Ref<Server> server;

      const int resolutionX;
      const int resolutionY;
      const double aspectRatio;

      const double nearClippingDist;
      const double farClippingDist;

      const bool useMipmapedTextures;
```

```
      CoordSystem3 viewCoordSystem;
      double fieldOfView;
      double depthOfRotation;

      Ref<const Viewpoint> boundViewpoint;

      Vector2 viewPlane;

      RenderConfig renderConfig;

      bool wireframeMode;
      bool enableTexture;

      bool separateSpecularColorMode;
      bool lightModelLocalViewerMode;

      int numberOfEnabledLights;
      bool tracePointingRay;
      Vector3 pointingRay;
      double pointingRayLength;
      vector<const GroupingNode *> hitGroupThread;
      int hitGroupThreadLowerLevel;
      int level; // Nesting level of groups, 0 is root group
      Ref<const GroupingNode> sensorGroup;
      bool pointingDeviceDrag;
      bool sensorGroupActive;
      bool ableToActivate;
      Vector3 originalHitIntersection;

      bool forcedShapeCacheUpdate;

      void setupViewpoint();
      Viewpoint *findBoundViewpoint_group
      (const GroupingNode *, stack<const Transform *> *transformPath=0);
      Viewpoint *findBoundViewpoint_child
      (ChildNode *, stack<const Transform *> *transformPath=0);

      void setupLight(bool headLight, bool showLightSources);
      void lightPointLight(const PointLight *, int &, bool);
      void lightSpotLight(const SpotLight *, int &, bool);
      void lightDispatchLightNode(const LightNode *, int &, bool);
      void lightTransform(const Transform *, int &, bool);
      void lightSwitch(const Switch *, int &, bool);
      void lightDispatchGroupingNode(const GroupingNode *, int &, bool);
      void lightGroupingNode(const GroupingNode *, int &, bool);

      void dispatchChildNode(const ChildNode *);
      void drawGroupingNode(const GroupingNode *, bool);
      void dispatchGroupingNode(const GroupingNode *, bool);
      void drawTransform(const Transform *, bool);
      const GeometryNode *drawShape(const ShapeNode *);
      void leaveSensorGroup();
      void enterSensorGroup();
      void activateSensorGroup(const Math::Ray3 &);
      void deactivateSensorGroup();
      void sensorTrack(const Math::Ray3 &);
```



```
      struct Hit
      {
        const GeometryNode *node;
        int where;
        double distance;
        double localDistance; // Makes no sense as it is identical to '
             distance'.
        Math::Ray3 localRay;
      };

      Hit hit;

      struct ShapeCache;
    };
  }
}

#endif // ARCHON_X3D_VIEW_H
```

Listing: **archon/x3d/server/view.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#include <math.h>
#include <typeinfo>
#include <iostream>
#include <map>

#include <GL/gl.h>
#include <GL/glu.h>

#include <archon/math/vector.H>
#include <archon/util/image.H>
#include <archon/util/text.H> // Tempo

#include <archon/x3d/server/field_type.H>
#include <archon/x3d/server/load.H>
#include <archon/x3d/server/server.H>
#include <archon/x3d/server/view.H>

using namespace std;
```

```
namespace Archon
{
  using namespace Math;

  namespace X3D
  {
    Viewer::Viewer(Ref<Server> server,
                   int resolutionX, int resolutionY,
                   bool useMipmapedTextures,
                   string initialViewpointName,
                   const CoordSystem3 &viewCoordSystem,
                   double fieldOfView,
                   double depthOfRotation,
                   const Vector4 &backgroundColor,
                   int subdivisionX, int subdivisionY,
                   bool showNormals,
                   bool wireframeMode, bool enableTexture):
      server(server),
      resolutionX(resolutionX), resolutionY(resolutionY),
      aspectRatio(double(resolutionX)/resolutionY),
      nearClippingDist(1), farClippingDist(2000),
      useMipmapedTextures(useMipmapedTextures),
      viewCoordSystem(viewCoordSystem),
      fieldOfView(fieldOfView),
      depthOfRotation(depthOfRotation),
      renderConfig(subdivisionX, subdivisionY, showNormals, depthOfRotation
           /10),
      wireframeMode(wireframeMode), enableTexture(enableTexture),
      separateSpecularColorMode(false),
      lightModelLocalViewerMode(false),
      numberOfEnabledLights(0),
      sensorGroup(0), pointingDeviceDrag(false), sensorGroupActive(false),
      ableToActivate(true),
      forcedShapeCacheUpdate(false)
    {
      GLint param;
      glGetIntegerv(GL_MAX_LIGHTS, &param);
      maxLights = param;

      glViewport(0, 0, resolutionX, resolutionY);

      glClearColor(backgroundColor[0], backgroundColor[1],
                   backgroundColor[2], backgroundColor[3]);
      glDepthFunc(GL_LESS);
      glEnable(GL_DEPTH_TEST);

      // Disable all light sources
      for(unsigned i=0; i<maxLights; ++i) glDisable(GL_LIGHT0+i);

      if(server->getRootContext())
      {
        // Bind to the specified viewpoint
        if(!initialViewpointName.empty())
        {
          if(!bindViewpoint(initialViewpointName))
            server->getLogger()->log("WARNING: No such viewpoint '" +
                                     initialViewpointName + "'");
```



```
        }

        // Bind to the first viewpoint if any
        if(!boundViewpoint)
          boundViewpoint =
            findBoundViewpoint_group(server->getRootContext()->
                                     getRootGroup().get());
      }
    }

    bool Viewer::bindViewpoint(string name)
    {
      if(!server->getRootContext())
        ARCHON_THROW1(InternalException,
                      "Viewer::bindViewpoint: No root scene");

      Ref<Node> n = server->getRootContext()->lookupNode(name);
      if(Viewpoint *v = dynamic_cast<Viewpoint *>(n.get()))
      {
        boundViewpoint = v;
        return true;
      }
      return false;
    }

    /**
     * \todo The 'centerOfRotation' property of the Viewpoint is not
     * properly handled. We currently simplify this to a
     * 'depthOfRotation' since it makes more sense in connection with
     * our virtual trackball.
     */
    void Viewer::setupViewpoint()
    {
      if(boundViewpoint)
      {
        stack<const Transform *> transformPath;
        if(!server->getRootContextNoLock())
          ARCHON_THROW1(InternalException,
                        "Viewer::setupViewpoint: No root scene");
        GroupingNode *g = server->getRootContextNoLock()->getRootGroupNoLock
             ().get();
        if(findBoundViewpoint_group(g, &transformPath))
        {
          viewCoordSystem = CoordSystem3::identity();
          while(!transformPath.empty())
          {
            const Transform *t = transformPath.top();
            transformPath.pop();
            viewCoordSystem.translate(t->getTranslation());
            viewCoordSystem.translate(t->getCenter());
            viewCoordSystem.basis.rotate(t->getRotation());
            viewCoordSystem.basis.rotate(t->getScaleOrientation());
            viewCoordSystem.basis.scale(t->getScale());
            viewCoordSystem.basis.rotate(-t->getScaleOrientation());
            viewCoordSystem.translate(-t->getCenter());
          }
          viewCoordSystem.translate(boundViewpoint->getPosition());
          viewCoordSystem.basis.rotate(boundViewpoint->getOrientation());
```

```
      fieldOfView = boundViewpoint->getFieldOfView();

      depthOfRotation = dist(boundViewpoint->getPosition(),
                             boundViewpoint->getCenterOfRotation());
    }
  }

  const double viewPlaneRadius = nearClippingDist*tan(fieldOfView/2);
  if(aspectRatio > 1)
  {
    viewPlane[0] = viewPlaneRadius * aspectRatio;
    viewPlane[1] = viewPlaneRadius;
  }
  else
  {
    viewPlane[0] = viewPlaneRadius;
    viewPlane[1] = viewPlaneRadius / aspectRatio;
  }
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glFrustum(-viewPlane[0], viewPlane[0], -viewPlane[1], viewPlane[1],
            nearClippingDist, farClippingDist);
  glMatrixMode(GL_MODELVIEW);

  CoordSystem3 invViewCoordSystem;
  invViewCoordSystem.setInverseOf(viewCoordSystem);
  GLdouble m[16];
  invViewCoordSystem.setOpenGLMatrix(m);
  glLoadMatrixd(m);
}

Viewpoint *Viewer::findBoundViewpoint_child(ChildNode *c,
                                             stack<const Transform *> *
                                                  transformPath)
{
  if(Viewpoint *v = dynamic_cast<Viewpoint *>(c))
  {
    if(!boundViewpoint ||
       boundViewpoint.get() == v) return v;
    return 0;
  }

  if(GroupingNode *g =
     dynamic_cast<GroupingNode *>(c))
  {
    if(Viewpoint *v = findBoundViewpoint_group(g, transformPath))
    {
      if(transformPath)
        if(Transform *t = dynamic_cast<Transform *>(g))
          transformPath->push(t);
      return v;
    }
    return 0;
  }

  return 0;
}
```



```
Viewpoint *Viewer::findBoundViewpoint_group(const GroupingNode *g,
                                            stack<const Transform *> *
                                                 transformPath)
{
  GroupingNode::const_iterator c=g->begin();

  if(const Switch *s = dynamic_cast<const Switch *>(g))
  {
    int i = s->getWhichChoise();
    if(i<0) return 0;
    c += i;
    if(c >= g->end()) return 0;
    return findBoundViewpoint_child(c->get(), transformPath);
  }

  while(c!=g->end())
  {
    if(Viewpoint *v = findBoundViewpoint_child(c->get(), transformPath))
      return v;
    ++c;
  }

  return 0;
}


/**
 * \bug We fail to check for GL_MAX_LIGHTS with the fatal
 * behaviour of possibly overwriting completely different
 * settings.
 *
 * \todo Maybe we should make the HeadLight a SpotLight.
 */
void Viewer::setupLight(bool headLight, bool showLightSources)
{
  int nextLight = 0;

  if(showLightSources)
  {
    glPointSize(8);
    glEnable(GL_POINT_SMOOTH);
    glDisable(GL_LIGHTING);
    glDisable(GL_TEXTURE_2D);
  }

  if(!server->getRootContextNoLock())
    ARCHON_THROW1(InternalException,
                  "Viewer::setupLight: No root scene");
  GroupingNode *g = server->getRootContextNoLock()->getRootGroupNoLock().
       get();
  lightGroupingNode(g, nextLight, showLightSources);

  if(headLight)
  {
    int i = GL_LIGHT0 + nextLight++;

    GLfloat v[4];
```

```
      v[3] = 1;

      v[0] = 0.2;
      v[1] = 0.2;
      v[2] = 0.2;
      glLightfv(i, GL_AMBIENT, v);

      v[0] = 0.8;
      v[1] = 0.8;
      v[2] = 0.8;
      glLightfv(i, GL_DIFFUSE, v);

      v[0] = 0.8;
      v[1] = 0.8;
      v[2] = 0.8;
      glLightfv(i, GL_SPECULAR, v);

      v[0] = viewCoordSystem.origin[0];
      v[1] = viewCoordSystem.origin[1];
      v[2] = viewCoordSystem.origin[2];
      glLightfv(i, GL_POSITION, v);

      glLightf(i, GL_SPOT_CUTOFF, 180);

      glLightf(i, GL_CONSTANT_ATTENUATION,  1);
      glLightf(i, GL_LINEAR_ATTENUATION,    0);
      glLightf(i, GL_QUADRATIC_ATTENUATION, 0);

      glEnable(i);
    }

    for(int i=nextLight; i<numberOfEnabledLights; ++i)
      glDisable(GL_LIGHT0+i);

    numberOfEnabledLights = nextLight;
  }

  /**
   * \todo What should the specular color be set to? It is not
   * specified by the LightNode's.
   *
   * \todo The radius of the LightNode is not taken into
   * account. How should it be?
   */
  void Viewer::lightPointLight(const PointLight *l, int &nextLight, bool
       show)
  {
    if(l->getOn())
    {
      int i = GL_LIGHT0 + nextLight++;

      GLfloat v[4];
      v[3] = 1;

      v[0] = l->getColor()[0] * l->getAmbientIntensity();
      v[1] = l->getColor()[1] * l->getAmbientIntensity();
      v[2] = l->getColor()[2] * l->getAmbientIntensity();
      glLightfv(i, GL_AMBIENT, v);
```



```
      v[0] = l->getColor()[0] * l->getIntensity();
      v[1] = l->getColor()[1] * l->getIntensity();
      v[2] = l->getColor()[2] * l->getIntensity();
      glLightfv(i, GL_DIFFUSE, v);
      glLightfv(i, GL_SPECULAR, v);

      v[0] = l->getLocation()[0];
      v[1] = l->getLocation()[1];
      v[2] = l->getLocation()[2];
      glLightfv(i, GL_POSITION, v);

      glLightf(i, GL_SPOT_CUTOFF, 180);

      glLightf(i, GL_CONSTANT_ATTENUATION,  l->getAttenuation()[0]);
      glLightf(i, GL_LINEAR_ATTENUATION,    l->getAttenuation()[1]);
      glLightf(i, GL_QUADRATIC_ATTENUATION, l->getAttenuation()[2]);

      glEnable(i);
    }

    if(show)
    {
      glColor3d(l->getColor()[0], l->getColor()[1], l->getColor()[2]);
      glBegin(GL_POINTS);
      glVertex3d(l->getLocation()[0], l->getLocation()[1], l->getLocation
           ()[2]);
      glEnd();
    }
  }

  /**
   * \todo What should the specular color be set to? It is not
   * specified by the LightNode's.
   *
   * \todo The radius of the LightNode is not taken into
   * account. How should it be?
   *
   * To compute the spot exponent in the range [0, 128] from a beam
   * angle in the range [0, 2*pi], we choose a certain threshold
   * close to 1 and say that the beam is fully intensive if only the
   * true intensity is above the threshold. This gives us
   * something to work with:
   *
   *   pow(cos(beamWidth), spotExponent) = threshold  <=>
   *
   *   spotExponent = log(threshold) / log(cos(beamWidth))
   *
   * A threshold value of 3/4 seems to be sensible. Try to draw the
   * cosine function in various powers and you'll see that it is (I
   * hope).
   */
  void Viewer::lightSpotLight(const SpotLight *l, int &nextLight, bool show
       )
  {
    if(l->getOn())
    {
      int i = GL_LIGHT0 + nextLight++;
```

```
    GLfloat v[4];
    v[3] = 1;

    v[0] = l->getColor()[0] * l->getAmbientIntensity();
    v[1] = l->getColor()[1] * l->getAmbientIntensity();
    v[2] = l->getColor()[2] * l->getAmbientIntensity();
    glLightfv(i, GL_AMBIENT, v);

    v[0] = l->getColor()[0] * l->getIntensity();
    v[1] = l->getColor()[1] * l->getIntensity();
    v[2] = l->getColor()[2] * l->getIntensity();
    glLightfv(i, GL_DIFFUSE, v);
    glLightfv(i, GL_SPECULAR, v);

    v[0] = l->getLocation()[0];
    v[1] = l->getLocation()[1];
    v[2] = l->getLocation()[2];
    glLightfv(i, GL_POSITION, v);

    v[0] = l->getDirection()[0];
    v[1] = l->getDirection()[1];
    v[2] = l->getDirection()[2];
    glLightfv(i, GL_SPOT_DIRECTION, v);

    const double threshold = 0.75;
    float spotExponent =
      l->getBeamWidth() >= l->getCutOffAngle() ? 0 :
      l->getBeamWidth() <= acos(pow(threshold, 1.0/128)) ? 128 :
      log(threshold)/log(cos(l->getBeamWidth()));
    glLightf(i, GL_SPOT_EXPONENT, spotExponent);

    glLightf(i, GL_SPOT_CUTOFF, l->getCutOffAngle()/M_PI*180);

    glLightf(i, GL_CONSTANT_ATTENUATION,  l->getAttenuation()[0]);
    glLightf(i, GL_LINEAR_ATTENUATION,    l->getAttenuation()[1]);
    glLightf(i, GL_QUADRATIC_ATTENUATION, l->getAttenuation()[2]);

    glEnable(i);
  }

  if(show)
  {
    glColor3d(l->getColor()[0], l->getColor()[1], l->getColor()[2]);
    glBegin(GL_POINTS);
    glVertex3d(l->getLocation()[0], l->getLocation()[1], l->getLocation
         ()[2]);
    glEnd();
  }
}

void Viewer::lightDispatchLightNode(const LightNode *l, int &nextLight,
     bool show)
{
  if(const PointLight *p = dynamic_cast<const PointLight *>(l))
    lightPointLight(p, nextLight, show);
  else if(const SpotLight *p = dynamic_cast<const SpotLight *>(l))
    lightSpotLight(p, nextLight, show);
```



```
}

void Viewer::lightTransform(const Transform *t, int &nextLight, bool show
     )
{
  glPushMatrix();

  glTranslatef(t->getTranslation()[0],
               t->getTranslation()[1],
               t->getTranslation()[2]);
  glTranslatef(t->getCenter()[0],
               t->getCenter()[1],
               t->getCenter()[2]);
  glRotatef(t->getRotation().angle/M_PI*180,
            t->getRotation().normalizedAxis[0],
            t->getRotation().normalizedAxis[1],
            t->getRotation().normalizedAxis[2]);
  glRotatef(t->getScaleOrientation().angle/M_PI*180,
            t->getScaleOrientation().normalizedAxis[0],
            t->getScaleOrientation().normalizedAxis[1],
            t->getScaleOrientation().normalizedAxis[2]);
  glScalef(t->getScale()[0],
           t->getScale()[1],
           t->getScale()[2]);
  glRotatef(t->getScaleOrientation().angle/M_PI*180*-1,
            t->getScaleOrientation().normalizedAxis[0],
            t->getScaleOrientation().normalizedAxis[1],
            t->getScaleOrientation().normalizedAxis[2]);
  glTranslatef(t->getCenter()[0]*-1,
               t->getCenter()[1]*-1,
               t->getCenter()[2]*-1);

  lightGroupingNode(t, nextLight, show);

  glPopMatrix();
}

void Viewer::lightDispatchGroupingNode(const GroupingNode *g, int &
     nextLight, bool show)
{
  if(const Transform *t = dynamic_cast<const Transform *>(g))
    lightTransform(t, nextLight, show);
  else if(const Switch *s = dynamic_cast<const Switch *>(g))
    lightSwitch(s, nextLight, show);
  else lightGroupingNode(g, nextLight, show);
}

void Viewer::lightSwitch(const Switch *s, int &nextLight, bool show)
{
  GroupingNode::const_iterator c=s->begin();
  int i = s->getWhichChoise();
  if(i < 0) return;
  c += i;
  if(c >= s->end()) return;
  if(const LightNode *l = dynamic_cast<const LightNode *>(c->get()))
    lightDispatchLightNode(l, nextLight, show);
  else if(const GroupingNode *g2 = dynamic_cast<const GroupingNode *>(c->
       get()))
```

```
    lightDispatchGroupingNode(g2, nextLight, show);
}

void Viewer::lightGroupingNode(const GroupingNode *g, int &nextLight,
     bool show)
{
  GroupingNode::const_iterator c=g->begin();
  while(c!=g->end())
  {
    if(const LightNode *l = dynamic_cast<const LightNode *>(c->get()))
      lightDispatchLightNode(l, nextLight, show);
    else if(const GroupingNode *g2 = dynamic_cast<const GroupingNode *>(c
         ->get()))
      lightDispatchGroupingNode(g2, nextLight, show);
    else if(const SubContextNode *s = dynamic_cast<const SubContextNode
         *>(c->get()))
    {
      Ref<ExecutionContext> d = s->getSubContextNoLock();
      if(d)
      {
        Ref<Group> h = d->getRootGroupNoLock();
        if(h) lightGroupingNode(h.get(), nextLight, show);
      }
    }
    ++c;
  }
}



struct TexUsage
{
  GLuint textureId;
  int useCount;

  TexUsage(): useCount(0) {}
};

static map<Loader::Contents::Id, TexUsage> texMap;

struct Viewer::ShapeCache: public Shape::DrawCache
{
  GLuint textureId;
  GLuint callListId;

  const Shape *shape;
  const GeometryNode *geometry;

  Time calllistStamp;

  const FieldBase *geometryField;

  Loader::Contents::Id contentsId;

  ShapeCache(const Shape *s): shape(s), geometry(0)
  {
    geometryField = Shape::type->lookupField("geometry");
    if(!geometryField)
```



```
      ARCHON_THROW1(InternalException,
                    "Could not find Shape.geometry field");
  }

  static vector<ShapeCache *> &getDisposedShapeCaches()
  {
    static vector<ShapeCache *> disposedShapeCaches;
    return disposedShapeCaches;
  }

  static void cleanUp()
  {
    for(unsigned i=0; i<getDisposedShapeCaches().size(); ++i)
      delete getDisposedShapeCaches()[i];
    getDisposedShapeCaches().clear();
  }

  void dispose()
  {
    getDisposedShapeCaches().push_back(this);
  }

  bool updateTexture(const Viewer *viewer)
  {
    ImageTexture *imageTexture = 0;
    Ref<const Loader::ImageContents> imageContents;
    const AppearanceNode *appearance = shape->getAppearance().get();
    if(appearance)
    {
      const Appearance *a = dynamic_cast<const Appearance *>(appearance);
      if(!a) ARCHON_THROW1(InternalException,
                           "'" + appearance->getType()->getName() +
                           "' is not supported yet");

      const TextureNode *textureNode = a->getTexture().get();
      if(textureNode)
      {
        imageTexture = dynamic_cast<ImageTexture *>
          (const_cast<TextureNode *>(textureNode));
        if(!imageTexture)
          ARCHON_THROW1(InternalException,
                        "'" + textureNode->getType()->getName() +
                        "' is not supported yet");

        Ref<const Loader::Contents> c = imageTexture->getContents();
        imageContents = dynamic_cast<const Loader::ImageContents *>(c.get
            ());
        if(c && !imageContents)
          ARCHON_THROW1(InternalException,
                        "Viewer::ShapeCache::updateTexture: Got "
                        "non-image contents object");
      }
    }

    Loader::Contents::Id newContentsId;
    if(imageContents) newContentsId = imageContents->getId();

    if(newContentsId == contentsId) return false;
```

```
// Remove old texture
if(contentsId)
{
  TexUsage &texUsage = texMap[contentsId];
  if(--texUsage.useCount == 0)
  {
    glDeleteTextures(1, &texUsage.textureId); // Ups - what if
                                              // the new
                                              // texture is
                                              // the same
                                              // texture?
    texMap.erase(contentsId);
  }
}

contentsId = newContentsId;

if(!contentsId) return true;

// Check for existing?
TexUsage &texUsage = texMap[contentsId];
if(texUsage.useCount++ == 0)
{
  // Inserted (new URI)

  const Image *image = &imageContents->image;
  int colorMode;
  switch(image->getComponentSpecifier())
  {
  case Utilities::Image::components_l:    colorMode = GL_LUMINANCE;
       break;
  case Utilities::Image::components_la:   colorMode =
       GL_LUMINANCE_ALPHA; break;
  case Utilities::Image::components_rgb:  colorMode = GL_RGB;
       break;
  case Utilities::Image::components_rgba: colorMode = GL_RGBA;
       break;
  default:
    ARCHON_THROW1(Viewer::InternalException,
                  "Unsuported color components");
  }

  int componentType;
  switch(image->getBitsPerComponent())
  {
  case 8:  componentType = GL_UNSIGNED_BYTE;  break;
  case 16: componentType = GL_UNSIGNED_SHORT; break;
  default:
    ARCHON_THROW1(Viewer::InternalException,
                  "Unsuported color component width");
  }

  glGenTextures(1, &textureId);
  texUsage.textureId = textureId;
  glBindTexture(GL_TEXTURE_2D, textureId);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
                  imageTexture->getRepeatS() ? GL_REPEAT : GL_CLAMP);
```



```
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
                    imageTexture->getRepeatT() ? GL_REPEAT : GL_CLAMP);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                    viewer->useMipmapedTextures ?
                    GL_NEAREST_MIPMAP_NEAREST : GL_LINEAR);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

    if(viewer->useMipmapedTextures)
      gluBuild2DMipmaps(GL_TEXTURE_2D, colorMode, image->getWidth(),
                        image->getHeight(), colorMode, componentType,
                        image->getDataPtr());
    else glTexImage2D(GL_TEXTURE_2D, 0, colorMode, image->getWidth(),
                      image->getHeight(), 0, colorMode, componentType,
                      image->getDataPtr());
  }
  else
  {
    textureId = texUsage.textureId;
  }

  return true;
}

bool updateCalllist()
{
  const GeometryNode *g = shape->getGeometry().get();
  if(!geometry && g) callListId = glGenLists(1);
  if(geometry && !g) glDeleteLists(callListId, 1);
  geometry = g;
  return geometryField->changedSince(shape, calllistStamp);
}

/**
 * \return true if calllist must be re-made.
 */
bool update(const Viewer *viewer)
{
  bool u = updateTexture(viewer) | updateCalllist();

  if(u) calllistStamp = viewer->server->getNextTimeStamp();

  return u;
}

~ShapeCache()
{{
  if(geometry) glDeleteLists(callListId, 1);
  if(contentsId)
  {
    TexUsage &texUsage = texMap[contentsId];
    if(--texUsage.useCount == 0)
    {
      glDeleteTextures(1, &texUsage.textureId);
      texMap.erase(contentsId);
    }
  }
}}
```

```
};

const GeometryNode *Viewer::drawShape(const ShapeNode *shapeNode)
{
  const Shape *shape = dynamic_cast<const Shape *>(shapeNode);
  if(!shape)
    ARCHON_THROW1(InternalException,
                  "'" + shapeNode->getType()->getName() +
                  "' is not supported yet");

  if(!shape->drawCache)
    const_cast<Shape *>(shape)->drawCache = new ShapeCache(shape);
  ShapeCache *cache = static_cast<ShapeCache *>(shape->drawCache);

  const bool update = cache->update(this) || forcedShapeCacheUpdate;
  if(!cache->geometry) return 0;

  const AppearanceNode *appearanceNode = shape->getAppearance().get();
  const Appearance *a = dynamic_cast<const Appearance *>(appearanceNode);
  if(appearanceNode && !a)
    ARCHON_THROW1(InternalException,
                  "'" + appearanceNode->getType()->getName() +
                  "' is not supported yet");

  const TextureNode *textureNode = a ? a->getTexture().get() : 0;
  ImageTexture *imageTexture=0;
  if(textureNode)
    imageTexture = dynamic_cast<ImageTexture *>
      (const_cast<TextureNode *>(textureNode));

  const TextureTransformNode *textureTransformNode =
    a ? a->getTextureTransform().get() : 0;
  const TextureTransform *t =
    dynamic_cast<const TextureTransform *>(textureTransformNode);
  if(textureTransformNode && !t)
    ARCHON_THROW1(InternalException,
                  "'" + textureTransformNode->getType()->getName() +
                  "' is not supported yet");

  const MaterialNode *materialNode = a ? a->getMaterial().get() : 0;
  const Material *material = dynamic_cast<const Material *>(materialNode
      );
  if(materialNode && !material)
    ARCHON_THROW1(InternalException,
                  "'" + materialNode->getType()->getName() +
                  "' is not supported yet");
  if(material)
  {
    // Lit object
    GLenum face = GL_FRONT_AND_BACK;
    GLfloat c[4];

    double alpha = 1 - material->getTransparency();
    c[3]=alpha;
    glEnable(GL_LIGHTING);

    c[0]=material->getDiffuseColor()[0];
    c[1]=material->getDiffuseColor()[1];
```



```
    c[2]=material->getDiffuseColor()[2];
    glMaterialfv(face, GL_DIFFUSE, c);

    c[0] *= material->getAmbientIntensity();
    c[1] *= material->getAmbientIntensity();
    c[2] *= material->getAmbientIntensity();
    glMaterialfv(face, GL_AMBIENT, c);

    c[0] = material->getSpecularColor()[0];
    c[1] = material->getSpecularColor()[1];
    c[2] = material->getSpecularColor()[2];
    glMaterialfv(face, GL_SPECULAR, c);

    glMaterialf(face, GL_SHININESS, material->getShininess()*128);

    c[0] = material->getEmissiveColor()[0];
    c[1] = material->getEmissiveColor()[1];
    c[2] = material->getEmissiveColor()[2];
    glMaterialfv(face, GL_EMISSION, c);

    glColor3d(material->getDiffuseColor()[0],
              material->getDiffuseColor()[1],
              material->getDiffuseColor()[2]);
  }
  else
  {
    // Unlit object
    glDisable(GL_LIGHTING);
    glColor3d(1,1,1);
  }

  const bool texture = cache->contentsId;

  bool texStack = false;

  if(texture && enableTexture && !wireframeMode)
  {
    if(t)
    {
      glMatrixMode(GL_TEXTURE);
      glPushMatrix();

      glLoadIdentity();
      glTranslatef(t->getCenter()[0],
                   t->getCenter()[1],
                   0);
      glScalef(t->getScale()[0],
               t->getScale()[1],
               0);
      glRotatef(t->getRotation()/M_PI*180,
                0, 0, 1);
      glTranslatef(t->getCenter()[0]*-1,
                   t->getCenter()[1]*-1,
                   0);
      glTranslatef(t->getTranslation()[0],
                   t->getTranslation()[1],
                   0);
```

```
      glMatrixMode(GL_MODELVIEW);

      texStack = true;
    }

    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, cache->textureId);
    if(imageTexture)
    {
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, imageTexture->
           getRepeatS() ? GL_REPEAT : GL_CLAMP);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, imageTexture->
           getRepeatT() ? GL_REPEAT : GL_CLAMP);
    }
  }
  else glDisable(GL_TEXTURE_2D);

  const GeometryNode *g = cache->geometry;
  if(!update) glCallList(cache->callListId);
  else if(g)
  {
    glNewList(cache->callListId, GL_COMPILE_AND_EXECUTE);
    g->render(texture, true, shape, &renderConfig);
    glEndList();
  }

  if(texStack)
  {
    glMatrixMode(GL_TEXTURE);
    glPopMatrix();
    glMatrixMode(GL_MODELVIEW);
  }

  return g;
}

void Viewer::drawTransform(const Transform *t, bool sensor)
{
  glPushMatrix();

  glTranslatef(t->getTranslation()[0],
               t->getTranslation()[1],
               t->getTranslation()[2]);
  glTranslatef(t->getCenter()[0],
               t->getCenter()[1],
               t->getCenter()[2]);
  glRotatef(t->getRotation().angle/M_PI*180,
            t->getRotation().normalizedAxis[0],
            t->getRotation().normalizedAxis[1],
            t->getRotation().normalizedAxis[2]);
  glRotatef(t->getScaleOrientation().angle/M_PI*180,
            t->getScaleOrientation().normalizedAxis[0],
            t->getScaleOrientation().normalizedAxis[1],
            t->getScaleOrientation().normalizedAxis[2]);
  glScalef(t->getScale()[0],
           t->getScale()[1],
           t->getScale()[2]);
  glRotatef(t->getScaleOrientation().angle/M_PI*180*-1,
```



```
            t->getScaleOrientation().normalizedAxis[0],
            t->getScaleOrientation().normalizedAxis[1],
            t->getScaleOrientation().normalizedAxis[2]);
  glTranslatef(t->getCenter()[0]*-1,
               t->getCenter()[1]*-1,
               t->getCenter()[2]*-1);

  drawGroupingNode(t, sensor);

  glPopMatrix();
}

void Viewer::dispatchGroupingNode(const GroupingNode *g, bool sensor)
{
  if(const Transform *t = dynamic_cast<const Transform *>(g))
    drawTransform(t, sensor);
  else drawGroupingNode(g, sensor);
}

void Viewer::drawGroupingNode(const GroupingNode *g, bool sensor)
{
  int numberOfDirectionalLights = 0;

  /*
   * Is this the group that introduces the first enabled sensors
   * for this branch?  If it is, we raise the flag 'sensor'.
   *
   * Also, setup all enabled directional light sources that are
   * direct children of this group.
   *
   * Note that if this is a Switch node then neither pointing
   * device sensors nor directional lights could have any
   * effect. This is because they affect only their siblings and
   * the children of their siblings, but since this is a Switch
   * node then at most one child can be active at one point in
   * time. Thus, there can be no siblings.
   */
  if(!dynamic_cast<const Switch *>(g))
  {
    GroupingNode::const_iterator c=g->begin();
    while(c!=g->end())
    {
      if(const PointingDeviceSensorNode *s =
         dynamic_cast<const PointingDeviceSensorNode *>(c->get()))
      {
        if(!sensor && tracePointingRay && s->getEnabled())
        {
          sensor = true;
        }
      }
      else if(const DirectionalLight *l =
              dynamic_cast<const DirectionalLight *>(c->get()))
      {
        if(l->getOn())
        {
          ++numberOfDirectionalLights;
          int i = GL_LIGHT0 + numberOfEnabledLights++;
```

```
        GLfloat v[4];
        v[3] = 1;

        v[0] = l->getColor()[0] * l->getAmbientIntensity();
        v[1] = l->getColor()[1] * l->getAmbientIntensity();
        v[2] = l->getColor()[2] * l->getAmbientIntensity();
        glLightfv(i, GL_AMBIENT, v);

        v[0] = l->getColor()[0] * l->getIntensity();
        v[1] = l->getColor()[1] * l->getIntensity();
        v[2] = l->getColor()[2] * l->getIntensity();
        glLightfv(i, GL_DIFFUSE, v);
        glLightfv(i, GL_SPECULAR, v);

        v[0] = -l->getDirection()[0];
        v[1] = -l->getDirection()[1];
        v[2] = -l->getDirection()[2];
        v[3] = 0; // Push the position to infinity
        glLightfv(i, GL_POSITION, v);

        glLightf(i, GL_SPOT_CUTOFF, 180);

        glEnable(i);
      }
    }

    ++c;
  }
}

// Recurse into childs of this group
vector<const GeometryNode *> geometryNodes;
GroupingNode::const_iterator c=g->begin();
++level;
if(const Switch *s = dynamic_cast<const Switch *>(g))
{
  int i = s->getWhichChoise();
  if(i>=0)
  {
    c += i;
    if(c < g->end())
    {
      if(const ShapeNode *s = dynamic_cast<const ShapeNode *>(c->get
          ()))
      {
        const GeometryNode *g = drawShape(s);
        if(tracePointingRay && g) geometryNodes.push_back(g);
      }
      else if(const SubContextNode *s = dynamic_cast<const
          SubContextNode *>(c->get()))
      {
        Ref<ExecutionContext> d = s->getSubContextNoLock();
        if(d)
        {
          Ref<Group> h = d->getRootGroupNoLock();
          if(h) dispatchGroupingNode(h.get(), sensor);
        }
      }
```



```
      else if(const GroupingNode *g2 = dynamic_cast<const GroupingNode
            *>(c->get()))
        dispatchGroupingNode(g2, sensor);
    }
  }
}
else while(c!=g->end())
{
  if(const ShapeNode *s = dynamic_cast<const ShapeNode *>(c->get()))
  {
    const GeometryNode *g = drawShape(s);
    if(tracePointingRay && g) geometryNodes.push_back(g);
  }
  else if(const SubContextNode *s = dynamic_cast<const SubContextNode
        *>(c->get()))
  {
    Ref<ExecutionContext> d = s->getSubContextNoLock();
    if(d)
    {
      Ref<Group> h = d->getRootGroupNoLock();
      if(h) dispatchGroupingNode(h.get(), sensor);
    }
  }
  else if(const GroupingNode *g2 = dynamic_cast<const GroupingNode *>(c
       ->get()))
    dispatchGroupingNode(g2, sensor);

  ++c;
}
--level;

// Turn off directional light sources belonging to this group
while(numberOfDirectionalLights--)
  glDisable(GL_LIGHT0 + --numberOfEnabledLights);

if(!geometryNodes.empty())
{
  // Fetch and invert current model view matrix
  double m[16];
  glGetDoublev(GL_MODELVIEW_MATRIX, m);
  CoordSystem3 modelview(m);
  CoordSystem3 invModelview;
  invModelview.setInverseOf(modelview);

  // Map pointing ray into local coordinates
  Math::Ray3 localRay;
  localRay.direction = pointingRay;
  invModelview.basis.map(localRay.direction);
  localRay.origin = localRay.direction;
  localRay.origin += invModelview.origin;

  // Ray collision detection with extracted shapes
  bool anyHits = false;
  for(unsigned i=0; i<geometryNodes.size(); ++i)
  {
    double dist;
    int where = geometryNodes[i]->intersect(localRay, dist);
    if(where && (dist < hit.distance || hit.distance < 0))
```

```
        {
          hit.distance = dist;
          if(sensor)
          {
            hit.node = geometryNodes[i];
            hit.where = where;
          }
          anyHits = true;
        }
      }

      if(anyHits)
      {
        if(sensor)
        {
          hit.localDistance = hit.distance;
          hit.localRay = localRay;
          hitGroupThread.clear();
          hitGroupThreadLowerLevel = level+1;
        }
        else hit.node = 0;
      }
    }

    if(hit.node && level < hitGroupThreadLowerLevel)
    {
      hitGroupThread.push_back(g);
      hitGroupThreadLowerLevel--;
    }
  }

  void Viewer::leaveSensorGroup()
  {
    Time t = server->getNextTimeStamp();
    GroupingNode::const_iterator c=sensorGroup->begin();
    while(c!=sensorGroup->end())
    {
      if(const TouchSensorNode *s = dynamic_cast<const TouchSensorNode *>(c
           ->get()))
        if(s->isOver)
        {
          const_cast<TouchSensorNode *>(s)->isOver = false;
          Event e(new SimpleValue<bool>(SFBool::type, s->isOver), t);
          s->isOverChanged.cascadeEvent(&e);
        }

      ++c;
    }
  }

  void Viewer::enterSensorGroup()
  {
    Time t = server->getNextTimeStamp();
    GroupingNode::const_iterator c=sensorGroup->begin();
    while(c!=sensorGroup->end())
    {
      if(const TouchSensorNode *s = dynamic_cast<const TouchSensorNode *>(c
           ->get()))
```



```
        if(s->getEnabled() && !s->isOver)
        {
          const_cast<TouchSensorNode *>(s)->isOver = true;
          Event e(new SimpleValue<bool>(SFBool::type, s->isOver), t);
          s->isOverChanged.cascadeEvent(&e);
        }

      ++c;
    }
  }

  void Viewer::activateSensorGroup(const Math::Ray3 &ray)
  {
    Time t = server->getNextTimeStamp();
    originalHitIntersection = ray.direction;
    originalHitIntersection *= hit.localDistance * hit.localRay.direction.
         length() / ray.direction.length();
    originalHitIntersection += ray.origin;

    GroupingNode::const_iterator c=sensorGroup->begin();
    while(c!=sensorGroup->end())
    {
      if(const PointingDeviceSensorNode *s =
         dynamic_cast<const PointingDeviceSensorNode *>(c->get()))
        if(s->getEnabled() && !s->isActive)
        {
          const_cast<PointingDeviceSensorNode *>(s)->isActive = true;
          Event e(new SimpleValue<bool>(SFBool::type, s->isActive), t);
          s->isActiveChanged.cascadeEvent(&e);
        }

      ++c;
    }
    sensorGroupActive = true;
  }

  void Viewer::deactivateSensorGroup()
  {
    Time t = server->getNextTimeStamp();
    GroupingNode::const_iterator c=sensorGroup->begin();
    while(c!=sensorGroup->end())
    {
      if(const PointingDeviceSensorNode *s =
         dynamic_cast<const PointingDeviceSensorNode *>(c->get()))
      {
        if(s->isActive)
        {
          const_cast<PointingDeviceSensorNode *>(s)->isActive = false;
          Event e(new SimpleValue<bool>(SFBool::type, s->isActive), t);
          s->isActiveChanged.cascadeEvent(&e);

          if(const TouchSensorNode *ts = dynamic_cast<const TouchSensorNode
               *>(s))
            if(ts->isOver)
            {
              Event e(new SimpleValue<Time>(SFTime::type, t), t);
              ts->touchTimeChanged.cascadeEvent(&e);
            }
```

```
      }
    }
    ++c;
  }
  pointingDeviceDrag = false;
  sensorGroupActive = false;
  ableToActivate = true;
}

void Viewer::sensorTrack(const Math::Ray3 &ray)
{
  Time t = server->getNextTimeStamp();

  // OBS: need to take care of ray.direction[2] == 0, this situation
  // is described in X3D sensor spec.

  const double dist = (originalHitIntersection[2] - ray.origin[2])/ray.
       direction[2];

  GroupingNode::const_iterator c=sensorGroup->begin();
  while(c!=sensorGroup->end())
  {
    if(const DragSensorNode *d = dynamic_cast<const DragSensorNode *>(c->
         get()))
    {
      if(d->enabled)
      {
        // trackPoint_changed
        const Vector3 trackPoint(ray.origin[0] + dist*ray.direction[0],
                                 ray.origin[1] + dist*ray.direction[1],
                                 originalHitIntersection[2]);
        const_cast<DragSensorNode *>(d)->trackPoint = trackPoint;
        Event e(new SimpleValue<Vector3>(SFVec3f::type, trackPoint), t);
        d->trackPointChanged.cascadeEvent(&e);

        // translation_changed (PlaneSensor)
        if(const PlaneSensor *p = dynamic_cast<const PlaneSensor *>(d))
        {
          const Vector3 translation(ray.origin[0] + dist*ray.direction
               [0],
                                    ray.origin[1] + dist*ray.direction
                                         [1],
                                    originalHitIntersection[2]);
          const_cast<PlaneSensor *>(p)->translation = p->translation +
               translation - originalHitIntersection;
          Event e(new SimpleValue<Vector3>(SFVec3f::type, p->translation
               ), t);
          p->translationChanged.cascadeEvent(&e);
        }
      }
    }
    ++c;
  }
  pointingDeviceDrag = true;
}

/**
 * \todo We must also reset fieldOfView and depthOfRotation.
```



```
 */
void Viewer::resetViewpoint(string n, const CoordSystem3 &s)
{
  if(!server->getRootContext())
    ARCHON_THROW1(InternalException,
                  "Viewer::resetViewpoint: No root scene");

  // Attempt to bind to the specified viewpoint
  if(!n.empty()) bindViewpoint(n);

  // Otherwise bind to the first viewpoint if any
  if(!boundViewpoint)
    boundViewpoint =
      findBoundViewpoint_group(server->getRootContext()->
                               getRootGroup().get());

  // Otherwise use the default view point
  if(!boundViewpoint) viewCoordSystem = s;
}




/**
 * \todo Event cascade time stamping assumes that each cascade
 * takes more than one nanosecond. Another sollution may be
 * needed.
 *
 * \return true if the pointing device isOver a sensor.
 */
bool Viewer::view(bool headLight, bool showLightSources,
                  bool sensorModeActive, bool pointingDeviceActive,
                  const Vector2 *newMousePosition)
{
  if(newMousePosition)
  {
    /**
     * The pointing ray (mouse ray) must originate on the viewplane
     * (near clipping plane) to prevent collision with geometry that
     * is so close to the eye that it cannot be seen (is clipped
     * away).
     *
     * This in turn means that the origin and direction of the
     * pointing ray are always identical. Because this fact can be
     * used to cut down work load we represent the pointing ray only
     * by the single vector that defines both the starting point and
     * the direction of the ray.
     */
    pointingRay[0] = viewPlane[0] * (2 * (*newMousePosition)[0] /
         resolutionX - 1);
    pointingRay[1] = viewPlane[1] * (-2 * (*newMousePosition)[1] /
         resolutionY + 1);
    pointingRay[2] = -nearClippingDist;
    pointingRayLength = pointingRay.length();
    tracePointingRay = true;
  }
  else
  {
```

```
  tracePointingRay = false;
}

hit.node = 0;
level = 0;
hitGroupThread.clear();
hit.distance = -1; // Force first hit to be closer that "previous".

glShadeModel(GL_SMOOTH);
glColorMaterial(GL_FRONT_AND_BACK, GL_DIFFUSE);
glEnable(GL_NORMALIZE);
glPolygonMode(GL_FRONT_AND_BACK, wireframeMode ? GL_LINE : GL_FILL);
glLightModeli(GL_LIGHT_MODEL_COLOR_CONTROL, separateSpecularColorMode
      ? GL_SEPARATE_SPECULAR_COLOR : GL_SINGLE_COLOR);
glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, lightModelLocalViewerMode
      ? 1 : 0);

Mutex::Lock lock(server->scenegraphMutex);

server->tick(); // Generate TimeSensor events

/*
 * This must be done by the same thread that createss
 * ShapeCaches since in involves OpenGL commands.
 */
ShapeCache::cleanUp();

ExecutionContext *rootContext = server->getRootContextNoLock().get();
if(!rootContext)
{
  forcedShapeCacheUpdate = false;
  return false;
}
GroupingNode *rootGroup = rootContext->getRootGroupNoLock().get();
if(!rootGroup)
{
  forcedShapeCacheUpdate = false;
  return false;
}

glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(-1, 1, -1, 1, 1, 1000-1);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslated(0, 0, -10);

setupViewpoint();

setupLight(headLight, showLightSources);

/*
 * Render the scene and on request 'tracePointingRay=true' trace
 * the pointing ray for intersection with sensor affected
 * geometry. Upon return 'hit.node!=0' will indicate that the
 * ray has hit a sensor affected gemetry node and a reference to
 * that node is stored in 'hit.node'. The rest of the fields of
 * 'hit' will be set to reflect that hit, and 'hitGroupThread'
```



```
 * will contain the path given in grouping nodes starting with
 * the one imediately enclosing 'hit.node' to the root group of
 * the scene.
 */
drawGroupingNode(rootGroup, false);

forcedShapeCacheUpdate = false;

if(!sensorModeActive)
{
  if(sensorGroup) leaveSensorGroup();
  if(sensorGroupActive) deactivateSensorGroup();
  sensorGroup = 0;
  return false;
}

// tracePointingRay - select mode (spacebar) - Pointing Ray has been
     cast
// hit.node - pointingRay has hit a node
// pointingDeviceActive - primary button on pointing device is active
// sensorGroupActive - sensor group is active (has been activated)
// return value - true/false (active mouse cursor mode/inactive mouse
     cursor mode)

if(hit.node)
{
  /*
   * Traverse the scene graph towards the root in search of the
   * right sensor group
   */
  Math::Ray3 ray(hit.localRay);
  for(unsigned i=0; i<hitGroupThread.size(); ++i)
  {
    Ref<const GroupingNode> g(hitGroupThread[i]);
    GroupingNode::const_iterator c=g.get()->begin();
    while(c!=g.get()->end())
    {
      if(const PointingDeviceSensorNode *s =
         dynamic_cast<const PointingDeviceSensorNode *>(c->get()))
        if(s->getEnabled())
        {
          if(dynamic_cast<const TouchSensorNode *>(c->get()))
          {
            if(g != sensorGroup)
            {
              if(sensorGroup) leaveSensorGroup();
              sensorGroup = g;
              enterSensorGroup();
              return true;
            }
            else
            {
              enterSensorGroup();
            }
          }

          if(pointingDeviceActive && ableToActivate)
          {
```

```
                if(g == sensorGroup)
                {
                  if(sensorGroupActive)
                    sensorTrack(ray);
                  activateSensorGroup(ray);
                  return true;
                }
              }
              else
              {
                if(sensorGroupActive)
                  deactivateSensorGroup();
                if(!pointingDeviceActive) ableToActivate = true;
              }
            }
            else
            {
              if(g == sensorGroup)
              {
                if(sensorGroup) leaveSensorGroup();
                if(sensorGroupActive) deactivateSensorGroup();
                sensorGroup = 0;
                return false;
              }
            }
          ++c;
        }

        if(const Transform *t = dynamic_cast<const Transform *>(g.get()))
        {
          CoordSystem3 s(CoordSystem3::identity());
          s.translate(t->getTranslation());
          s.translate(t->getCenter());
          s.basis.rotate(t->getRotation());
          s.basis.rotate(t->getScaleOrientation());
          s.basis.scale(t->getScale());
          s.basis.rotate(-t->getScaleOrientation());
          s.translate(-t->getCenter());

          s.map(ray.origin);
          s.basis.map(ray.direction);
        }
      }

      // If there were no sensors that wanted to deal with the collision
      return true;
    }
    else
    {
      if(!sensorGroupActive && pointingDeviceActive) ableToActivate = false
           ;
      if(tracePointingRay && sensorGroup) leaveSensorGroup();
      if(sensorGroup && sensorGroupActive && pointingDeviceActive &&
           pointingDeviceDrag)
      {
        return true;
      }
```



```
        if(tracePointingRay && sensorGroupActive && !pointingDeviceActive)
            deactivateSensorGroup();
        if(tracePointingRay && !pointingDeviceActive) ableToActivate = true;
        if(tracePointingRay && sensorGroup) sensorGroup = 0;
        if(tracePointingRay) return false;
        else return sensorGroup;
      }
    }
  }
}
```

Listing: **archon/x3d/server/x3d.H**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#ifndef ARCHON_X3D_X3D_H
#define ARCHON_X3D_X3D_H

/**
 * The proper order of component header file inclusion that satisfies
 * both inheritance and containment hierachies is:
 *
 * - x3d/core.H          Core component
 * - x3d/time.H          Time component
 * - x3d/network.H       Networking component
 * - x3d/group.H         Aggregation and transformation component
 * - x3d/geoprop.H       Geometric properties component
 * - x3d/texture.H       Texturing component (+AppearanceChildNode)
 * - x3d/appear.H        Appearance component (-AppearanceChildNode)
 * - x3d/geometry.H      Geometry component
 * - x3d/text.H          Text component
 * - x3d/light.H         Lighting component
 * - x3d/interpolation.H Interpolation component
 * - x3d/navigate.H      Navigation component
 * - x3d/mouse.H         Pointing device sensor component
 * - x3d/script.H        Scriptinh component
 * - x3d/aux.H           Auxiliary component
 *
 * Note that the "X3DAppearanceChildNode" had to be moved from the
 * Appearance component to the Texture component to satisfy both
 * inheritance and containment hierachies.
```

```
 */
#include <archon/x3d/server/core.H>
#include <archon/x3d/server/time.H>
#include <archon/x3d/server/network.H>
#include <archon/x3d/server/group.H>
#include <archon/x3d/server/geoprop.H>
#include <archon/x3d/server/texture.H>
#include <archon/x3d/server/appear.H>
#include <archon/x3d/server/geometry.H>
#include <archon/x3d/server/text.H>
#include <archon/x3d/server/light.H>
#include <archon/x3d/server/interpolation.H>
#include <archon/x3d/server/navigate.H>
#include <archon/x3d/server/mouse.H>
#include <archon/x3d/server/script.H>
#include <archon/x3d/server/aux.H>

namespace Archon
{
  namespace X3D
  {
    /**
     * This function must be called prior to any use of the X3D library
     */
    void initialize();
  }
}

#endif // ARCHON_X3D_X3D_H
```

Listing: **archon/x3d/server/x3d.C**

```
/*
 * This file is part of the "Archon" framework.
 * (http://files3d.sourceforge.net)
 *
 * Copyright © 2002 by Kristian Spangsege and Brian Kristiansen.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation under the terms of the GNU General Public License is
 * hereby granted. No representations are made about the suitability of
 * this software for any purpose. It is provided "as is" without express
 * or implied warranty. See the GNU General Public License
 * (http://www.gnu.org/copyleft/gpl.html) for more details.
 *
 * The characters in this file are ISO8859-1 encoded.
 *
 * The documentation in this file is in "Doxygen" style
 * (http://www.doxygen.org).
 */

#include <archon/x3d/server/x3d.H>

namespace Archon
{
  namespace X3D
  {
    extern void initializeFieldTypes();
```



```
    extern void initializeNode();
    extern void initializeCoreComponent();
    extern void initializeTimeComponent();
    extern void initializeNetworkComponent();
    extern void initializeGroupComponent();
    extern void initializeGeopropComponent();
    extern void initializeTextureComponent();
    extern void initializeAppearComponent();
    extern void initializeGeometryComponent();
    extern void initializeTextComponent();
    extern void initializeLightComponent();
    extern void initializeInterpolationComponent();
    extern void initializeNavigateComponent();
    extern void initializeMouseComponent();
    extern void initializeScriptComponent();
    extern void initializeAuxComponent();

    void initialize()
    {
      initializeFieldTypes();
      initializeNode();
      initializeCoreComponent();
      initializeTimeComponent();
      initializeNetworkComponent();
      initializeGroupComponent();
      initializeGeopropComponent();
      initializeTextureComponent();
      initializeAppearComponent();
      initializeGeometryComponent();
      initializeTextComponent();
      initializeLightComponent();
      initializeInterpolationComponent();
      initializeNavigateComponent();
      initializeMouseComponent();
      initializeScriptComponent();
      initializeAuxComponent();
    }
  }
}
```