Information technology — Computer graphics, image processing and environmental data representation — Extensible 3D (X3D) — Part 2: Scene access interface (SAI)

ISO/IEC 19775-2:2015

This document is ISO/IEC 19775-2:2015, Extensible 3D (X3D), Edition 3. The full title of this part of ISO/IEC 19775 is: *Information technology — Computer graphics, image processing and environmental data representation — Extensible 3D (X3D) — Part 2: Scene access interface (SAI)*.

The ***Foreword*** provides background on the standards process for X3D. The ***Introduction*** describes the purpose, design criteria, and characteristics of X3D. The following clauses define this part of ISO/IEC 19775:

1. ***Scope*** defines the problem area that X3D addresses.
2. ***Normative references*** lists the normative standards referenced in this part of ISO/IEC 19775.

3. ***Terms, definitions, acronyms and abbreviations*** contains the glossary of terminology used in this part of ISO/IEC 19775.
4. ***Concepts*** describes various fundamentals of the X3D scene access interface.
5. ***Data type reference*** defines the data types used by the application programmer interfaces.
6. ***Services reference*** defines the functionality which may be accessed through the application programmer interfaces.
7. ***Conformance and minimum support requirements*** describes the conformance requirements for X3D implementations.

There is one annex included in the specification:

A. ***VRML 97 scripting backwards compatibility*** describes the manner in which X3D Scripting can be used to provide backwards compatibility with VRML 97 scripting.

update to match X3D4 proposed 19775-1?

Extensible 3D (X3D)
Part 2: Scene access interface (SAI)

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form a specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 19775-2 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee 24, *Computer graphics, image processing and environmental data representation*, in collaboration with Web3D Consortium, Inc.

This third edition cancels and replaces the second edition (ISO/IEC 19775-2:2010), which has been technically revised.

ISO/IEC 19775 consists of the following parts, under the general title *Information technology — Computer graphics, image processing and environmental data representation — Extensible 3D (X3D):*

Part 1: Architecture and base components
Part 2: Scene access interface (SAI) (this part)

update to match X3D 19775-1 ?

Extensible 3D (X3D)
Part 2: Scene access interface

Introduction

Purpose

X3D is a file format and related access services for describing interactive 3D objects and worlds. X3D is designed to be used on the Internet, intranets, and local client systems. X3D is also intended to be a universal interchange format for integrated 3D graphics and multimedia. X3D may be used in a variety of application areas such as engineering and scientific visualization, multimedia presentations, entertainment and educational titles, web pages, and shared virtual worlds.

This part of ISO/IEC 19775 defines the scene access interface that can be used to interact with X3D worlds both from within the worlds or from external programs.

Information technology — Computer graphics, image processing and environmental data representation — Extensible 3D (X3D) — Part 2: Scene access interface (SAI)

1 Scope

This part of ISO/IEC 19775 specifies a standard set of services that are made available by a browser so that an author can access the scene graph while it is running. Such access is designed to support inspection and modification of the scene graph.

^ consistent programmatic

programmer working with an authored Model

Extensible 3D (X3D)
Part 2: Scene access interface (SAI)

2 Normative references

☐ check versions 19775-1

☐ check prose 19775-1

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

Identifier	Reference
I10646	ISO/IEC 10646, *Information technology — Universal Multiple-Octet Coded Character Set (UCS)*
I14496-1	ISO/IEC 14496-1:2010, *Information technology — Coding of audio-visual objects — Part 1: Systems*
I14772-1	ISO/IEC 14772-1:1997, *Information technology — Computer graphics and image processing — The Virtual Reality Modeling Language — Part 1: Functional specification and UTF-8 encoding*
I14772-2	ISO/IEC 14772-2:2004, *Information technology — Computer graphics and image processing — The Virtual Reality Modeling Language (VRML) — Part 2: External authoring interface (EAI)*
I19775-1	ISO/IEC 19775-1:2013 → 2022, *Information technology — Computer graphics, image processing and environmental data representation — Extensible 3D (X3D) — Part 1: Architecture and base components*
I19776-1	ISO/IEC 19776-1:2015, *Information technology — Computer graphics, image processing and environmental data representation — Extensible 3D (X3D) encodings — Part 1: Extensible Markup Language (XML) encoding (in preparation)*
I19776-2	ISO/IEC 19776-2:2015, *Information technology — Computer graphics, image processing and environmental data representation — Extensible 3D (X3D) encodings — Part 2: Classic VRML encoding (in preparation)*
I19776-3	ISO/IEC 19776-3:2015, *Information technology — Computer graphics, image processing and environmental data representation — Extensible 3D (X3D) encodings — Part 3: Compressed binary encoding (in preparation)*

I19777	ISO/IEC 19777-1:201x, *Information technology — Computer graphics, image processing and environmental data representation — Extensible 3D (X3D) language bindings — Part 1: ECMAScript (in preparation)* ISO/IEC 19777-2:201x, *Information technology — Computer graphics, image processing and environmental data representation — Extensible 3D (X3D) language bindings — Part 2: Java (in preparation)*
RFC4248	IETF RFC 4248, *The telnet URI Scheme*, Internet standards track protocol
W3CDOM2	*W3C Document Object Model (DOM) Level 2 Core Specification Version 1.0*

☑ check

☐ check

Additional programming language bindings and file encodings.

link all images to TOC or top of page

Extensible 3D (X3D)
Part 2: Scene access interface (SAI)

3 Terms, definitions, acronyms and abbreviations

For the purposes of this document, the terms, definitions, acronyms and abbreviations given in ISO/IEC 19775-1 and the following apply.

3.1 errors

browser event cascade

reasons for unsuccessful termination of a service

3.2 file

encoding. A representation of an X3D model in accordance with 19775-1 (and other references) in accordance with a given file format (e.g. XML, Classic VRML, JSON, etc

collection of related data stored on physical media or existing as a data stream or as data within a computer program

3.3 initializeOnly field

author-defined

field defined as part of a node definition whose value may only be specified at the time that the node is instantiated

3.4 input-capable field

either an inputOnly field or an inputOutput field

3.5 inputOnly field

field defined as part of a node definition which may only receive events

3.6 inputOuput field

field defined as part of a node definition that is capable of both receiving events and sending events

→ language binding. An implementation of the X3D Scene Access Interface in a given programming language (e.g. ECMAScript, Java, etc.)

3.7
now

present time as specified by the user's system clock

3.8
output-capable field

either an inputOutput field or an outputOnly field

3.9
outputOnly field

field defined as part of a node definition which may only send events

3.10
parameters

values passed into a service interface

3.11
public interface

formal definition of a node type in this part of ISO/IEC 19775

3.12
returns

values returned by an invocation of a service interface

3.13
run-time name scope

extent to which a name defined within an X3D file applies and is visible

Extensible 3D (X3D)
Part 2: Scene access interface (SAI)

4 Concepts

4.1 Introduction

awk

This clause describes key concepts in this part of ISO/IEC 19775. This includes describing the various components of the browser and how the interactions with the browser may be accomplished. It does not define what the individual interactions are. Those descriptions can be found in 6 Services reference.

Table 4.1 provides links to the major topics in this clause.

Table 4.1 — Topics

4.4.10 Statements

x30

x30

4.2 Overview

4.2.1 General

When a user wishes to interact with an X3D scene graph through use of custom code, either as a Script node as defined in *29 Scripting component* in ISO/IEC 19775-1 or from external applications, they shall use the Scene Authoring Interface (SAI) defined in this part of ISO/IEC 19775. This interface is a protocol for manipulating the X3D scene graph while not directly part of the scene graph itself.

This specification is aimed at providing a language-neutral representation of all actions that can be performed by an external application across this interface. Bindings to specific languages are defined in ISO/IEC 19777. The SAI forms a common interface that can be used either for manipulating the browser and the scene graph from either an external application or from inside the scene graph through the Script node. However, it is not possible for code written for an external application to be immediately usable as a script. The two environments have quite different requirements and abilities to access and interact with the scene graph. This specification provides a single, unified programmatic interface and constraints that depend on the environment in which the code finds itself.

Conceptually, the SAI allows five types of access into the X3D scene:

- accessing the functionality of the Browser;
- receive notifications of the actions of the Browser, such as encountering bad URLs, startup and shutdown;
- sending events to input-capable fields of nodes inside the scene;
- reading the last value sent from output-capable fields of nodes inside the scene; and
- getting notified when events change values of node fields inside the scene.

4.2.2 Compatibility with ISO/IEC 14772

If an X3D browser wishes to conform to ISO/IEC 14772-1, the browser shall support the event model and semantics defined in Annex A VRML scripting backwards compatibility in addition to the functionality specified in ISO/IEC 19775-1. Such support shall only be used when processing files that conform to ISO/IEC 14772-1.

If an X3D browser wishes to conform to ISO/IEC 14772-2 (EAI), the browser shall use the following rules to determine content validity:

a. If the user code accesses the browser through the EAI, only VRML files as specified in ISO/IEC 14772-2 shall be loaded. It shall be an error to process files that conform to this part of ISO/IEC 19775 if the user code is accessing the browser through the EAI.
b. If the user code accesses the browser through the capabilities provided by the external interactions defined in this part of ISO/IEC 19775, only X3D files as defined in ISO/IEC 19775-1 shall be loaded. If shall be an error to process files that conform to ISO/IEC 14772-1 if the user code is accessing the browser through the SAI.

4.3 Binding and protocol dependencies

Implementation dependence is defined in terms of the language binding or protocol encoding of the services defined in this specification. If a service is defined to be implementation dependent, it is a requirement of each binding and encoding to specify how that service is to be implemented, if at all.

Bindings and encodings to these services may define their own implementation dependent parts within that specification.

4.4 Interface constructs

4.4.1 Overview

There are four main data collections in an X3D browser that can be accessed using the services of the SAI: The browser, meta data about the currently loaded scene, nodes within the scene graph, and fields within nodes. The definition and specifications are framed in terms of services. An X3D browser exposes a set of services that allow external applications to interact with it. In order to describe these concepts, a number of terms are defined.

4.4.2 User code

Any code that makes use of the services defined in this part of ISO/IEC 19775 is considered to be user code. User code may exist either within the scene graph or external to the browser. It shall only use the services provided by this part and no browser implementation specific services. In addition, these services are not designed for, nor intended to be used for, writing native node extensions to a specific browser. A browser may provide its own proprietary programmatic interfaces to implement native extensions that are not part of this specification. If code uses proprietary extensions, it shall not be considered user code for the purposes of this part of ISO/IEC 19775.

4.4.3 Containing node

A containing node is the node in the scene graph that is responsible for representing user code that wishes to take part in internal interactions (see 4.8.3 Internal interactions). The life cycle of user code shall be governed entirely by the containing node. When the containing node becomes live, the user code becomes live. When the containing node is removed and is no longer considered live as defined in *4.4.2.5 Object life cycle* in ISO/IEC 19775-1, the user code contained by that node shall be terminated. User code cannot prolong the lifetime of the containing node by keeping a reference to its containing node. The browser is the final arbiter of when the containing node is no longer live.

There is no requirement for there to be a one-to-one mapping between a containing node and its user code. Language bindings may permit one instance of user code to be shared between multiple instances of a containing node.

4.4.4 Application

An application is the external process that is not implicitly part of the X3D browser. This application makes some form of connection to the X3D browser along which requests are made of the browser. The application does not exist as part of the X3D browser as defined in *Figure 4.1* in ISO/IEC 19775-1 nor forms part of the execution model defined in *4.4.8.3 Execution model* in ISO/IEC 19775-1. An application may reside on another machine from the X3D browser.

An application may be responsible for creating a new browser instance that is embedded within that application or attaching itself to an already running instance of a browser (for example, an applet on a web page).

4.4.5 Session

A session defines the life of a single connection between the user code and the X3D browser. It is possible for a single browser to be servicing multiple sessions simultaneously (for example, multiple script nodes in the one scene).

A single application may contain a number of separate sessions to multiple browsers, but a single script node shall not. Multiple simultaneous sessions between external applications and multiple X3D browsers are permissible. However, individual implementations may place some restrictions on such multiple simultaneous sessions.

A session is not an implementable part of this specification. It is purely a conceptual mechanism by which the user can make requests for services. It may exist prior to any connection being established between a browser and external application or is established simultaneously with the request for a browser connection.

4.4.6 Browser

The browser is the basic encapsulation mechanism for an active X3D scene graph (that is one where time is progressing, not as a file stored on disk). As it contains the entire scene graph, it also provides a minimal core set of capabilities for dynamically manipulating that scene graph at a coarse level. This scene graph may contain at most one active LayerSet node and that node shall be a root node of the scene graph (see *4.3.2 Root nodes* of ISO/IEC 19775-1). Any other LayerSet node contained in a scene imported using an Inline node (or through any other means) shall be ignored.

A user may have many X3D browsers running simultaneously on their machine. Therefore, each browser shall be represented by a unique identifier within that session. This identifier is required to be identical for multiple requests of a single browser instance. This ~~is to~~ enable two applications that have access to the one browser instance to share information in an unambiguous way.

Any action that requires use of the browser functionality shall identify the service request with a browser identifier.

4.4.7 Scene

A scene represents a single X3D scene graph and all information about that scene graph. The scene is the programmatic equivalent of an X3D file. It may contain nodes, routes, proto declarations, imports and exports and all information a valid X3D file may contain. A browser may contain one or more scenes at any given time. For example one scene uses an Inline node to include another scene.

A scene is not required to be live or running in the browser. A user may construct a new scene that is not attached to a browser instance and then programmatically fill in information such as nodes and routes. This scene may then be passed directly to a utility program such as a pretty printer for publishing a source file or used to replace the current scene in the browser.

4.4.8 Node and node lifecycle

The smallest unit of interaction with the elements in the scene graph is the node. A node is an instance of one of the X3D nodes that are defined in ISO/IEC 19775-1. A node can be removed as a unit from the scene graph, stored, and then re-inserted at another position at some later time in the same session without detrimental effect.

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 external application make service requests of the browser with that shared data.

Most operations in the SAI begin by obtaining a reference to a node. There are multiple ways to gain a reference to a node. It may be named using the DEF construct and fetched using the appropriate service or it may be obtained by walking the scene graph from some arbitrary parent node. Once a reference is obtained, all fields of that node may be accessed, but not necessarily read or written, including initializeOnly fields. Since an inputOutput field implicitly may be both read and written, these are accessible using the field name or with the *set_* and *_changed* modifiers.

A node reference undergoes a lifecycle during which different capabilities are available. The lifecycle can be expressed as:

1. Creation: The node is first instantiated by the browser internals with all field values set to defaults
2. Setup: Field values are changed from the default value to the defined initial values where required
3. Realized: The node is participating in the scene graph and/or scripting
4. Disposed: The node is no longer part of a scene graph and no remaining references to it exist at the scripting level.

Field access for reading and writing is dependent on the state of the node. The states and capabilities are defined in Table 4.2.

Table 4.2 — Permitted field access capabilities during the node lifecycle.

Field type	Creation	Setup	Realized	Disposed
initializeOnly field	None	readable/writable	None	None

inputOnly field	None	None	writable	None
outputOnly field	None	None	readable	None
inputOutput field	None	readable/writable	readable/writable	None

The transition from setup to realized states may be either implicit or explicit. A service request exists so that the user may make a formal notification that setup is now finished and the node can complete whatever internal construction is required. The transition may be implicit due to the user's actions. At the point the user does anything with the node reference other than set the field values, the node shall transition to the realized state.

EXAMPLE The user creates a Box node, sets the size field, creates a Shape node, and then immediately adds the Box to the Shape node. This set of actions shall result in the state of the Box node changing to Realized, while leaving the Shape node in the setup state.

Node identifiers may also be used to represent an empty node. An empty SFNode or MFNode field value is represented by a NULL value. For empty MFNode fields, the count of available nodes shall be zero.

4.4.9 Field

Individual fields are defined within nodes. While it is not possible to directly manipulate a node, a field is the method of direct manipulation of individual properties as indicated in Table 4.2.

It is not possible to directly manipulate a node's properties as entities separate from the node itself (i.e., fields do not exist outside their containing nodes).

The field type and access type of individual fields is specified by Part 1 of ISO/IEC 19775. A field is assigned a field identifier. This is non-unique and requires a node identifier plus the field identifier to specify a particular field with which to interact. When accessing a field, the user shall be given the whole identifier to the field. All fields are implicitly treated as being both readable and writable by the service definition. Flags are used to indicate whether that field can be read or written at that point in time (and dependent on the node's state in the lifecycle as described in Table 4.2). This state may change over time as the node progresses through its lifecycle. For example, an initializeOnly field of a non-live node may be writable, but once that node is inserted into the scene graph, it shall no longer be writable. This is to aid authoring tools and users that wish to programmatically construct a scene around a third party browser.

Fields may be read or written at any time during the course of the session. User code may register and unregister to receive notification of when values of the field changes. During the registration process the user code can supply a token that will be returned along with the data value of the event. This token can be used by the user code to uniquely identify this event in cases where events are

not implicitly unique. The token is not required to be passed along with the service request and may be kept as part of the internals of the implementation on the application interface.

Any output-capable field of a node to which the application has a reference can be read. The value read is the last value sent by that field or the default value for that field type if no event has ever been sent. The data read is specific to the field type of that field and is formatted appropriate to the language or protocol used.

4.4.10 Execution context

An execution context is the run-time semantic equivalent to a name scope described in *4.4.7 Run-time name scope* in ISO/IEC 19775-1. It provides a way of containing and firewalling internal interaction code in such a manner as to represent the same restrictions that a name space provides in the file format. For example, when a script inside a Proto instance adds a ROUTE, the route is added to the internals of the proto and not to the general scene.

A scene is a derived type of execution context. When the internal interaction requests the current execution context, a scene object is returned. The user code may then check to see if the execution context is an instance of a full scene and behave appropriately by casting up to the derived type, if available.

4.5 Events

4.5.1 Concepts

Any transient data is carried around the X3D scene graph through the use of events. The application may register to receive events from the X3D scene graph, and may initiate new events. Events are considered transient and generated only at the time when the specific action occurs. Events shall not be stored, and have the delivery deferred to parties who have not expressed interest in the event at the time it occurred.

EXAMPLE An application that connects to a browser after the world has loaded shall not be delivered an Initialize event.

4.5.2 Internal to browser

An application may write a value to a field or read a value from a field. This value does not become an event until that value is internally represented and time stamped within the X3D browser. The border of the browser to the application is where an event stops. Events cannot exist externally from the X3D browser; that is, the application cannot be inserted in the middle of an event cascade. The application may be notified of events, initiate new events, but cannot process and pass on events while holding up processing of the current timestamp event cascades within the browser when it is notified of an event. It is permissible to log events for analytic purposes.

An event is not generated until a cascade is created. If an internal interaction directly writes to an output-capable field of another node, no event is generated and therefore does not form part of the event cascade. If the internal interaction writes to a input-capable field of the containing node, an event is formed with the written value, if the field is output-capable and is the subject of a ROUTE to somewhere else.

4.5.3 Browser to external application

4.5.3.1 Overview

The browser may directly communicate to external applications with its own set of events. These events are used to indicate the status of the browser or of some asynchronous problem. The number and type of events available shall be implementation dependent. At a minimum, the following events shall be provided in all implementations of this specification.

Event delivery from the browser to the external application shall be guaranteed.

4.5.3.2 Initialized

The initialize event is used to indicate that the browser has had a scene loaded where it has run through the initialization process (where the browser has loaded the world and just before it is about to issue its first time-related event). At this point in time, node identifiers shall be available from the getNode service of the scene (see 6.4.7 getNode).

The initialize event shall be generated immediately at the browser and delivered to the application. The event is considered to be asynchronous. That is, the delivery of the event (and any implementation dependent acknowledgement scheme) shall not delay the browser in starting the execution model evaluation.

4.5.3.3 Shutdown

The shutdown event is used to indicate that the browser is about to stop running the current scene. This may occur under a number of different conditions:

a. the scene is being replaced (see 6.3.12 replaceWorld and 6.3.14 loadURL),
b. the browser itself is exiting, or
c. the client application has disposed of its connection to the browser (see 6.3.25 dispose).

The shutdown event shall be generated immediately at the browser and delivered to the application. The event is considered to be asynchronous. That is, the delivery of the event (and any implementation dependent acknowledgement scheme) shall not delay the browser in halting the execution model evaluation and closing down of the browser resources except where needed to ensure the delivery of the event to the application.

4.5.3.4 No URLs available

The SAI_BROWSER_URL_ERROR event is used to notify the application that the browser was not able to load any of the URL/URNs in one of the asynchronous invocations of the `loadURL` service (See 6.3.14 loadURL). This indicates that no valid content was able to be loaded or invoked from any of the URLs specified in this call. Other calls that may involve other asynchronous loads such as `replaceWorld` (see 6.3.12 replaceWorld) and `createX3DFromString` and `createX3DFromStream` (see 6.3.16 createX3DFromString, 6.3.17 createX3DFromStream) may also use this event to indicate loading problems for any X3DUrlNode as specified in *9.3.2 X3DUrlObject* in ISO/IEC 19775-1 such as Inlines, textures and EXTERNPROTOs, although it is not required.

4.5.3.5 Connection lost

The connection lost error is used to notify the application that the underlying implementation has lost the connection between the browser and the application that would result in service requests not being able to be honoured. An example would be a TCP network connection timing out or other similar problem.

An implementation may delay sending an event that the connection has been lost if it implements an automatic reconnection attempt. It shall only be sent at the point where it is deemed no longer possible to connect to the browser. There shall be no requirement for the implementation to attempt to re-establish the connection after this event has been generated or to attempt any form of automatic reconnection capability.

4.6 Identifiers

What constitutes an identifier is implementation dependent. In some cases it may be more efficient to represent a node identifier as the entire node which includes all field information. Requests for field information are then made on the local node. In other implementations an identifier may be only a simple integer. The job of ensuring unique identifiers is the sole responsibility of the browser such that applications may share data within reasonable constraints of the environment. The constraints on that environment may be specified as part of the individual implementation.

It is not considered reasonable that two applications using different service implementations be able to exchange data outside of the browser environment.

4.7 Relative URLs

9.2.2 Relative URLs in ISO/IEC 19775-1 specifies the rules for dealing with relative URLs within a browser environment. The declaring file shall be defined as the base URL of the currently loaded world in the browser. The currently loaded world can be obtained by a request of the `getWorldURL` service (see 6.4.6 getWorldURL). In the case where a browser does not yet have an X3D file loaded,

the base document directory shall be taken to be the current working directory of the browser. Where the browser is part of a web page, the current working directory shall be treated as the base URL of the page in which the web browser is embedded.

When nested relative URLs are generated (such as an EXTERNPROTO containing a reference to a script file) the top level relative URL base is then resolved in accordance with ISO/IEC 19775-1.

4.8 Execution model

4.8.1 Overview of the interaction types

Because the SAI fulfills the role of the programmatic interface for both external applications and scripts, the execution model is capable of working in both situations. Although the API calls are identical for both situations, the run-time evaluation of each service request may be different.

EXAMPLE Servicing a field-changed notification in an internal script shall pause the current event cascade; for an external application it shall not.

This specification defines two types of interactions in which services may participate: internal (*i.e.*, a script) and external (*i.e.*, an application).

4.8.2 Event model evaluation order

Scripting code allows the user to modify the scene graph with custom behaviours. For consistent effects, the evaluation order defined in *4.4.8.3 Execution model* in ISO/IEC 19775-1 is expanded to include the service interactions allowed by a script. When internal interaction code is provided, the following order shall be used to evaluate all aspects of the event model.

1. Update camera based on currently bound Viewpoint's position and orientation.
2. Evaluate sensor input.
3. Gather external input from buffer and pass to nodes.
4. Call the `prepareEvents` script service for all live script nodes in the scene.
5. Evaluate routes.
6. Call the `shutdown` service on scripts that have received set_url events or are being removed from the scene
7. Generate final events for any sensors removed from the scene.
8. Add/remove any routes required by an invocation of the `dynamicRouteHandling` service request as defined in 6.4.17 dynamicRouteHandling from any script execution in step 6.
9. Call the `eventsProcessed` script service for scripts that have sent events generated in step 6.
10. Call the `initialize` service for newly loaded internal interaction code.

11. If any events were generated from steps 5 through 10, go to step 5 and continue until complete for the current event cascade.

If an internal interaction registers any form of callback or listener functionality with objects defined by this specification, those callbacks are made at the time the change occurs.

EXAMPLE A user code in Script A issues an event on an outputOnly field during the initialize service handling, and another piece of user code in Script B has a listener service on that eventOut. The listener in Script B would be fired immediately after the user code in Script A exits and returns control to the browser core.

4.8.3 Internal interactions

4.8.3.1 Permitted interactions

An internal interaction is when the user code and containing node form part of the X3D scene graph. These nodes are subject to and participate in the event cascade evaluation. Internal interactions may occur in the middle of the event cascade as a direct result of receiving an event, and may generate one or more output events in response. These events shall continue in the current cascade. When the output events are generated from asynchronous script evaluation or from some other process not directly related to processing of the current event cascade as defined in *29.2.4 EventsProcessed()* in ISO/IEC 19775-1, a completely new event cascade is started.

An X3DScriptNode type as specified in *29.3.1 X3DScriptNode* in ISO/IEC 19775-1 specifies a containing node although other node types may also be defined in the future. A browser shall only permit internal interactions by user code that is referenced from an X3DScriptNode type (or other future defined containing node type). If the user code is referenced from any other node type, it shall consider the code as an external interaction and act accordingly.

Internal interactions also permit direct interaction with fields of other nodes, or some browser operations without participating in the event cascade. This action shall only be allowed dependent on the value of the *directOutput* field setting of the containing *X3DScriptNode* node. The definitions of when this behaviour is permitted is defined in *29 Scripting component* in ISO/IEC 19775-1.

Because the user code is considered to be held inside the containing node, the view of the node's fields are reversed to the normal situation. An inputOnly field is a readable field, not writable; an outputOnly field is a writable field, not readable; and an initializeOnly as well as an inputOutput field are both readable and writable. Contrast this with an external node that is not the containing node in Table 4.3. This resembles the type of access that any other built-in or native extension node may have.

Table 4.3 — Permitted field interactions of a live node

Access Type	Containing Node	External Node
initializeOnly	readable/writable	no access

inputOnly	readable	writable
inputOutput	readable/writable	readable/writable
outputOnly	writable	readable

4.8.3.2 Browser interactions

Internal interactions are permitted with the browser. Because the code lies inside the current scene, they are only permitted a limited set of the full browser services. The services clause outlines which services will be available to internal actions and which are off limits.

During the initialization phase of the internal action, a script shall be given a reference to the browser that is appropriate for its interactions. This reference shall remain constant throughout the lifetime of the script while its containing node is considered live.

4.8.3.3 Responding to events

The purpose of an internal interaction is to respond to events, provide some processing and, optionally, also generate new output for the containing node. It may, also, provide asynchronous output that does not correspond to any input through the use of one or more threads. Generating output is considered in 4.8.3.4 Updating the scene graph. 4.8.3.4 describes the issues of responding to input resulting from an event cascade that sends an event to one or more output-capable fields of the containing node.

In order to respond to events, user code registers interest in the appropriate field(s) of the containing node. Interest may be registered with all field access types except outputOnly fields of the containing node. Once the containing node has completed its initialization phase, any time that one of the fields of the containing node receives an event the user code shall be notified of this through the notification mechanism provided by the language-specific bindings. The browser may choose to either immediately notify the user code or to batch a number of events together and provide a deferred notification. In either case, the browser shall ensure that all events for that timestamp are delivered during the event cascade for that timestamp and not at some later time. The browser shall also obey the containing node's *mustEvaluate* field directive as specified in *29.4.1 Script* in ISO/IEC 19775-1 when deciding whether to defer or immediately notify.

Upon notification, the browser shall not process any more events in the containing node's event cascade until processing has returned from the user code (although this does permit other event cascades to continue simultaneous processing). Values written to fields of other nodes and the input-capable fields of the containing node shall not be passed on to the destination node before the user code has relinquished control.

NOTE This does not exclude a browser implementation from delivering multiple events simultaneously to the user code if there are parallel event cascades being evaluated (for example a browser running on a multi-CPU machine where parallel event cascades can be evaluated and result in two cascades

delivering events to the containing node simultaneously). The writer of the user code should be aware of, and take appropriate precautions for, an event cascade occurring in parallel.

When the browser has determined that the cascade or cascades are completed, the browser may then call the containing node's `eventProcessed()` method as defined in *29.2.4 EventsProcessed()* in ISO/IEC 19775-1. The user code is also notified of this situation at which point the user code may then choose to perform extra evaluation and generate more output. User code for internal notifications has no way of determining when the current rendered frame has finished and the next frame begins.

4.8.3.4 Updating the scene graph

User code may choose to generate output in addition to receive inputs. For internal interactions, user code is not required to generate output in response to inputs. User code may asynchronously generate output or write directly to other nodes at any time that the containing node is considered live, within certain restrictions that are outlined in 4.8.3.3 Responding to events and 4.8.3.5 Asynchronous actions.

User code has two options for influencing the scene graph; it may write to the output-capable fields of the containing node and have the values be subject to the usual event cascade, or it may directly write to the input-capable fields of another node to which the containing node and user code already has a reference. User code may write to the containing node's fields at any time and in accordance with the access rules defined in Table 4.2. Internal interactions with other nodes shall be subject to the rules defined by the containing node's *directOutput* field as specified in *29.2.6 Scripts with direct outputs* in ISO/IEC 19775-1.

There are two special cases of user code not being permitted to make changes to fields of the containing node. The two fields of the X3DScriptNode abstract type *mustEvaluate* and *directOutput* are considered special and the user code shall not be permitted to modify these values at runtime. User code may read these values. Scripts can be self-modifying, thus, if the containing node is also derived from the X3DUrlObject abstract type (see *9.3.2 X3DUrlObject* in ISO/IEC 19775-1), it may choose to change its own URL fields, thereby replacing the current user code with new user code.

4.8.3.5 Asynchronous actions

User code in some languages is allowed to operate using asynchronous threads of execution. These allow user code to run without the need for direct stimulus from the browser. A typical use of this situation is to monitor a network connection for changes to be made to the scene graph. This requires the use of internal interactions that are not created as a direct result of field changes being received by the user code.

Internal interactions are only permitted at the times specified by 4.8.3.7 User code lifecycle. The browser shall generate an error if user code attempts to make internal interactions at any other time. The `prepareEvents` service (see 6.11.4.1 prepareEvents), if defined by the user code, allows user code to perform completely asynchronous changes to the scene graph, at known points in time,

without the need to clock the script using a TimeSensor or other node. This is in contrast to the eventsProcessed service (see 6.11.4.2 eventsProcessed), which is only called after the containing node has had to process field changes.

In addition, when user code registers for one of the listener services, the callbacks associated with this are considered asynchronous actions. User code operating during this period shall not be permitted to make modifications to the scene graph.

4.8.3.6 Monitoring changes in the scene graph

The services definition for fields allows user code to register interest in the output of fields of other nodes. Internal interaction code shall not be permitted to register interest in field change information. If the user code wishes to be informed of field change information, it shall use the existing route mechanism and the appropriate scene services to add a route between the field of interest and an input-capable field of the containing node.

Internal user code shall only be permitted to register interest in the output-capable fields of other nodes when the containing node's *directOutput* field is set to TRUE. It shall be an error to allow user code to register interest in outputs if this value is set to FALSE.

4.8.3.7 User code lifecycle

4.8.3.7.1 Overview

The lifetime of user code may be shorter for many reasons, such as the download time to fetch the code from a remote site or other user code changing the URL of the containing node to replace the current user code. However, the lifecycle of user code follows the same basic principles of the containing node. It has the same phases and undergoes similar transitions.

The lifecycle of the containing node is defined in *4.4.2.5 Object life cycle* in ISO/IEC 19775-1.

4.8.3.7.2 Setup

It is assumed that there will be some delay, however small, between when the containing node is initialized and when the user code will go through its initialization phase. While the containing node may already have finished the initialization phase and be in the running phase, the user code may not have started or may be processing its initialization.

During the initialization phase, internal interaction code is given all the information it needs for the rest of its lifecycle. The first step of the initialization phase is the instantiation of the user code. At instantiation, user code has no information about its containing environment or the containing node. During this time the user code may elect to set up any resources it requires, such as threads,

network connections or any other permitted actions of the containing environment (see 4.8.3.9 Execution environment and security).

After instantiation, the user code receives notification of resources needed to function within the internal interaction environment. It is given the identifier of the containing browser, the list of fields of a node (excluding any special fields) and the identifier of the containing node (needed so that user code may add and remove routes to its containing node). User code shall not make any service requests with the one exception of printing messages during this period. If it does, the browser shall generate an error.

As the last step of the initialization phase, the user code shall have its `initialize` service called (see 6.11.3.3 initialize). At this point user code is free to make use of all the services available to internal interactions.

NOTE This would be a good time for the user code to register for change notifications of the containing node fields or to perform external tasks like binding a particular viewpoint.

4.8.3.7.3 Realization

During the runtime phase, user code is subject to the requirements of this clause for receiving, sending and monitoring events as well as the X3D execution model.

User code is restricted about when it may make modifications to the scene graph. It shall only be permissible to make modifications in response to prompts from the browser. The permitted times shall be defined as:

1. During the pre-event cascade processing service request `prepareEvents` (see 6.11.4.1 prepareEvents);
2. In response to a change notification for an input-capable field of the containing node; and
3. During the post-event processing service request through the `eventsProcessed` service request (see 6.11.4.2 eventsProcessed).

It shall be an error for user code to make a service request at any time other than those where asynchronous interactions are permitted. Each service request in 6 Service reference defines whether user code is permitted to make asynchronous requests.

4.8.3.7.4 Disposal

User code will enter the shutdown phase when either the node is no longer live or the action of other user code has resulted in the user code being removed from the containing node (for example, changing the URL of a script node to point at new executable content).

Notification of the change to the shutdown phase shall be through the calling of the `shutdown` service request in the user code (see 6.11.5.1 shutdown). During this phase user code may set values to an output-capable field of the containing nodes or write final values directly to the input-capable fields of other nodes. At the end of the phase, the identifiers to the browser and containing fields shall be

considered as invalid. For example, if the user code contains a thread that continues to operate after the shutdown phase, it shall not be permitted to make modifications to the scene graph. To do so shall generate an error.

4.8.3.8 inputOutput fields and the containing node

The containing node is permitted to have fields with the inputOutput access type. Because an inputOutput field represents both an inputOnly and outputOnly field the user code may wish to write to the values, user code is subject to some special conditions in order to remain consistent with the core specification.

For the purposes of defining the allowable behaviour, the containing node and the user code are considered as decoupled, non-related entities. A notification of change in field value is a notification, and no more. Setting the value of a field that is defined as inputOutput is considered to be an instantaneous, atomic action. When the field is set, the value for both the input and output are set immediately. Then the notification to the user code is performed. Since the output of the inputOutput field has been set, any further attempt to change the value of the inputOutput field during the current timestamp is considered to be subject to *4.4.8.4 Loops* in ISO/IEC 19775-1. That is, if the user code receives an event notification for its containing node's inputOutput field, it cannot write another value to that same inputOutput field during the same timestamp because an output event has already been issued and the containing node is not permitted to issue another output event for the same field in the same timestamp.

In receiving events, the script shall only pass through the first event received by the containing node of the inputOutput field.

If the containing node has not yet received a change to the field during the current timestamp, the user code is permitted to write a value to the field. If a change is received to the field after the user code has modified it, only the inputOnly portion of the node is processed. A notification is sent to the user code, but the value of the field shall not be changed in accordance with the loop-breaking rule in *4.4.8.4 Loops* in ISO/IEC 19775-1.

4.8.3.9 Execution environment and security

All user code participating within a particular internal interaction environment is considered to operate within a single execution space. Code in this context is subject to the security settings of the containing browser's environment and also the language's operating environment. This permits user code in internal interactions to communicate through asynchronous mechanisms that are external to the X3D execution environment (i.e., route and event evaluation). Some language bindings may be subject to more restrictions than others. This is implementation independent.

EXAMPLE User code using the Java language bindings may operate in a web-browser sand box that does not allow network connections to any external server, while user code using user API bindings in an application may be given full access to the entire underlying operating system.

For security purposes, a browser may implement whatever schemes it feels necessary to ensure good security and to prevent content from undertaking

nefarious activities. Such activities may include virus-like modification of the user's computer or denial of service activities or any other activity deemed a security risk ~~on the day~~.

4.8.4 External interactions

4.8.4.1 Permitted interactions

User code that is interacting with a browser from an external perspective is considered to have complete control over the entire lifecycle ~~of not only~~ the scene graph ~~but also~~ and the browser. External interactions therefore have the full range of control over the browser.

Because an application is consider to be external to the browser, it does not have ~~intimate~~ knowledge of ~~the~~ internal state and therefore when actions may or may not be safe to make. Therefore, the external interactions are ~~defined to be~~ in an advisory capacity. An external interaction requests the browser make changes and then the browser shall decide exactly when it it safe to act on those requests. A browser shall honour all requests made, within the bounds of the individual services guideline outlined below.

An external application may also wish to monitor changes in nodes, fields and even the browser itself. The browser shall inform the external application of the changes, but shall do so in an asynchronous way. That is, any updates are considered to be notifications only, and shall not ~~hold up~~ the browser's internal evaluations. The result is that notifications may make it to the external application with some delay from when they happened within the browser.

EXAMPLE Delays may occur for data transfer between an external application sitting on a remote computer and a browser, due to transmission lags throughout the system.

4.8.4.2 Browser interactions

Browser interactions for external interactions include all the basic services provided to internal interactions. ~~In addition to this~~ a number of additional interactions are allowed. A single external application is permitted to interact with more than one browser at a time. It may also instruct multiple browsers to act ~~together~~ as a single entity or to work individually. The lifetime of the external application is independent of the browser.

4.8.4.3 Updating the scene graph

A characteristic of external applications is that they may make a lot of changes in bursts to the X3D browser. It is also possible that a single browser may have a number of applications connected to it, all making requests of the browser.

Events can be batched to aid in performance of the application (see 6.3.19 updateControl). The ~~mechanism~~ provided by this is a simple gate mechanism to hold all requests (`BeginUpdate`) to update the currently loaded world until the gate is released (`EndUpdate`).

When BeginUpdate is invoked, all requests to modify the contents of the current world are buffered and not passed to the browser. This buffering effects all requests to modify the current world including calls to loadURL and replaceWorld. Once a call to BeginUpdate has been made, any further BeginUpdate requests are ignored until the next call to EndUpdate at which time EndUpdate releases all of the currently buffered updates to the browser for processing.

If a modification service request is made on the scene after an endUpdate and before a BeginUpdate, it shall be passed to the scene immediately with the timestamp at the discretion of the browser.

BeginUpdate/EndUpdate requests shall be limited to the individual session. A request by one application to BeginUpdate shall only buffer the requests made by that application and not any others that may be connected to that same browser instance.

When EndUpdate is invoked, the following order of execution of requests shall be applied:

1. node setValues;
2. event cascade evaluation as defined in *4.4.8.3 Execution model* in ISO/IEC 19775-1.

The loadURL/replaceWorld service requests (see 6.3.14 loadURL and 6.3.12 replaceWorld) are not affected by the update control process. As soon as the browser receives these requests their execution is begun. The service definitions define the complete behaviour of these requests.

Buffered requests from the application shall be processed before processing any more requests either through another buffered queue or individual requests.

4.8.4.4 Monitoring changes in the scene graph

External interactions allow monitoring of any changes in the scene graph. Notifications of these changes shall be delivered in a timely manner and shall remain in the same sequence in which they are generated by the browser internals.

4.8.4.5 Synchronizing multiple applications

When multiple applications make requests of the browser, the requests shall be serviced in order of arrival time at the browser. The browser shall determine the arrival time. Buffered updates to the scene graph shall have their arrival time determined to be at the time that EndUpdate is requested. The arrival time is not necessarily the same as the timestamp at which the browser chooses to send events into the scene graph. The timestamp that the events are sent to the scene graph shall be determined by the browser but shall be no earlier than the time that EndUpdate is requested. The arrival time is used to sort out conflicting requests from multiple applications to ensure consistent results in the application of events in the correct order.

Should the browser determine that two requests arrive simultaneously the result is implementation dependent.

NOTE It is permissible for the external applications to send new values to a given inputOnly field simultaneously. For such situations the browser shall obey *4.4.8.5 Fan-in and fan-out* in ISO/IEC 19775-1. Authors can avoid non-deterministic behavior by ensuring that events occur in separate event cascades.

Should the browser receive a request to `loadURL` or `replaceWorld` while currently processing a similar request, the old request is immediately terminated and the new one begun. See 6.3.12 replaceWorld and 6.3.14 loadURL for more information.

4.8.5 Service guarantees

All requests for services shall be guaranteed to be honoured where the underlying implementation supports that service. Once the application has made a service request, that request shall be transmitted to the browser assuming that a connection is still available. That is, all communications are assumed to be reliable. Delivery is not guaranteed if the connection between the browser and application has been broken (for example, a TCP connection fails). Implementations shall define an error condition that notifies the user that the connection has failed for each service request. Also, the browser interface may include an event that provides asynchronous notification to the user of the failure.

Extensible 3D (X3D)
Part 2: Scene access interface (SAI)

5 Data type reference

5.1 Introduction and topics

5.1.1 Introduction

This clause describes the language independent data types used in the definition of 6 Services reference.

5.1.2 Topics

Table 5.1 specifies the topics for this clause.

Table 5.1 — Topics

5.1.3 Concepts

All data types in this clause are language binding independent. They represent single value information that is passed as parameters, return values or error conditions that can be generated through the external authoring interface. Each language binding shall define implementations of each of these data types.

These data types represent the specific implementation of each type; that is, how the X3D browser would represent them internally.

5.2 Data type definitions

5.2.1 SAIAction

SAIAction is a single value representing a qualifier for a more general service type. Each use of the SAIAction type shall define the range of acceptable values.

5.2.2 SAIBoolean

SAIBoolean represents a TRUE or FALSE value.

5.2.3 SAIBrowserApp

TODO consider Application

SAIBrowserApp is a data type that represents the complete browser application. This is different from 5.2.5 SAIBrowserRef because SAIBrowserApp defines the browser application itself whereas the SAIBrowserRef defines a reference to the standardized interface to the browser's functionality. The data type shall contain some method for obtaining an SAIBrowserRef.

5.2.4 SAIBrowserName

SAIBrowserName defines a representation of the name of the browser. If the browser implementation does not support this information, a NULL value is considered a legal representation of this data type.

5.2.5 SAIBrowserRef

TODO consider Reference

SAIBrowserRef represents a browser reference. This is a unique identifier per browser instance. Individual language bindings may place conditions on uniqueness allowing other methods for checking equivalent references to the same browser.

The browser concept is further defined in 4.4.6 Browser.

5.2.6 SAIBrowserVersion

information specific to the current implementation

SAIBrowserVersion defines a representation of the version of the browser. If the browser implementation does not support this information, a NULL value is considered a legal representation of this data type.

5.2.7 SAIComponentDeclaration

SAIComponentDeclaration defines all the information about a component and its declaration. It may be used to represent both the component information declared in a file, and the available components from the browser.

5.2.8 SAIComponent

SAIComponent specifies an identifier of a component when used in a request. Components consist of a name and a level and both are encapsulated in this identifier.

5.2.9 SAIEncoding

SAIEncoding specifies an identifier for an encoding type.

5.2.10 SAIExecutionContext

SAIExecutionContext is a data type for a representation of a subscene piece of information relating to the current name space as a run-time entity.

5.2.11 SAIFieldAccess

This data type defines the type of access that is permitted to a field. The valid values are initializeOnly, inputOnly, outputOnly, and inputOutput.

5.2.12 SAIFieldDeclaration

SAIFieldDeclaration represents the abstract declaration of a field for a node. The declaration is constant for all instances of that node and does not include the field value. It can be considered a wrapper data type that includes SAIFieldAccess, SAIFieldName and SAIFieldType data types.

5.2.13 SAIField

SAIField represents an identifier for a particular field of a node. It is guaranteed to be unique within the scope of an individual node reference. It is not guaranteed to be unique in terms of all field references generated. To uniquely define a field within the scene graph, a combination of node and field identifiers is needed.

The field concept is further defined in 4.4.9 Field.

5.2.14 SAIFieldName

SAIFieldName represents a name for a field.

5.2.15 SAIFieldType

SAIFieldType specifies the type of data a field represents. In some cases (where the field type represents an MFNode or SFNode), this field type may correspond to an SAINode. Valid types of the field are defined in *5 Field type reference* in ISO/IEC 19775-1.

5.2.16 SAIFieldValue

SAIFieldValue represents the value to be set or to be returned of an SAIFieldType in language specific terms. A table may be constructed mapping each SAIFieldType represented to at least one language specific entry. This data type contains an item of class SAIFieldType and an item of the field type specified by the value of the first item. All field types defined in *5 Field type reference* in ISO/IEC 19775-1 shall be supported.

5.2.17 SAIFrameRate

SAIFrameRate represents the rendering rate in frames per second that is currently being achieved by the browser.

5.2.18 SAILayerID

SAILayerID is an identifier of the target layer for an operation. The ordering of the layers is the ordinal position of the layer in the layers field of the LayerSet

node defined in 35.4.2 LayerSet in ISO/IEC 19775-1.

[verify]

5.2.19 SAILoadState

SAILoadState represents the load state of a node or EXTERNPROTO instance. The state shall be one of NOT_STARTED, IN_PROGRESS, COMPLETE or FAILED.

5.2.20 SAIMatrix

SAIMatrix specifies an identifier for a 3×3 or 4×4 matrix.

5.2.21 SAINavSpeed

SAINavSpeed represents the navigation speed of the user in base speed units.

5.2.22 SAINode

SAINode specifies an identifier for a node. Individual language bindings may place conditions on uniqueness allowing other methods for checking equivalent references to the same node. The node reference is not required to be part of the active scene graph.

or empty node list

A NULL value is a legal value for this data type. It is used to indicate that no node identifier is to be used. For example, for the field service `setValue` (see 6.7.6 setValue), a value of NULL on an SFNode field type is used to clear the node that may have previously been set as the value. Simply a value of [ ] is used to clear an MFNode field type that may have had previous values -

The node concept is further defined in 4.4.8 Node.

5.2.23 SAINodeType

SAINodeType represents the type of a node.

5.2.24 SAIParameterList

Martin: can NULL be part of an MFNode list?

SAIParameterList is the abstract data type used to represent a list of parameters that may be passed to a service request. Each language binding shall be required to define the exact listing of parameters and their mapping to language-specific types. This may be used to represent more than one list of parameters where a binding provides multiple overloaded implementations of a single service.

5.2.25 SAIProfileDeclaration

SAIProfileDeclaration specifies all the information about a profile and its declaration. It may be used to represent both the profile information declared in a file, and the available profiles from the browser.

5.2.26 SAIPropertyList

SAIPropertyList is an abstract data type defining a set of key/value pairs for the provision of properties.

5.2.27 SAIProtoDeclaration

SAIProtoDeclaration represents the declaration of a PROTO or EXTERNPROTO. It does not represent an instance created from the declaration. The declaration cannot be changed at runtime and is only constructed from a file, stream or string. This allows a browser to build optimized internal storage mechanisms that may not be traversable using normal scene graph traversal mechanisms. Each language binding shall define its own representation and methods for creating instances of the PROTO declaration.

5.2.28 SAIRequester

SAIRequester represents the identifier of a client application or part thereof that is requesting a service to be performed. Variables of this data type are usually used to identify a particular client piece of code that is interested in listening for changes in some information in either the scene graph or browser state that functions as a callback device.

5.2.29 SAIRoute

SAIRoute represents a ROUTE construct. A ROUTE is not a node in the scene graph and does not represent a concrete structure (see 4.4.8.2 Routes in ISO/IEC 19775-1).

5.2.30 SAIScene

SAIScene represents a complete world and all the information that defines it including nodes, routes, protos and exports. A scene may come from parsing of a file, stream or string, or be programmatically constructed through this API.

5.2.31 SAIScript

SAIScript represents the containing node for user code that performs internal interactions. It is an X3DNode object that exists as part of the X3D scene graph.

5.2.32 SAIScriptImplementation

SAIScriptImplementation is a marker data type used by user code to mark the entrance point for the user code execution. It shall provide the lifecycle methods that the browser calls during user code execution such as *initialize()* and *shutdown()*. It shall not be used as a parameter to, or return type of, any service definition.

5.2.33 SAIStream

SAIStream represents X3D content arriving continuously.

5.2.34 SAIString

SAIString represents a string formatted with the UTF-8 universal character set. (see ISO/IEC 10646).

5.2.35 SAIUnitDeclaration

SAIUnitDeclaration defines all the information about units and its declaration. It is used to represent the unit information declared for a file, either explicitly or by default.

5.2.36 SAIURL

SAIURL is a data type which references a single URL. The URL may be any valid URL representation but is usually defined as a human-readable string that conforms to 2.[RFC4248].

5.2.37 NULL

for nodes.

NULL represents the empty value. It contains no data or reference to any data type but serves as a valid value to return from a service when nothing can be returned and yet no error is generated.

Empty list (Component Unit Meta ROUTE etc.)

what about statements?

5.3 Error types

5.3.1 SAIError

This section defines the error types that may be generated in response to service requests. Errors are generated as synchronous values from a service request and returned as variables of type SAIError. These error types appear in the errors definition of a service request (see 6.1.3 Conventions used). A language binding shall define the representation for the SAIError data type and assign values for each of the errors defined below but may also define additional error data types to these.

5.3.2 SAI_BROWSER_UNAVAILABLE

This error indicates that the request to gain a reference to a SAIBrowserApp has failed. Examples may be that a network connection is down or that the type of reference required is not supported by the vendor-specific implementation of a language binding.

5.3.3 SAI_CONNECTION_ERROR

An error has occurred that resulted in the connection between the browser and external application becoming non-functional. Therefore, the service request

could not be executed. This is a different error condition from SAI_BROWSER_UNAVAILABLE as it assumes that a valid reference has already been obtained and the error occurred at a later time.

5.3.4 SAI_DISPOSED

The request made of the current SAINode, SAIField or SAIBrowserRef reference is being made to an object that has already been disposed prior to this service request.

5.3.5 SAI_IMPORTED_NODE

An operation was attempted that used an imported node when it is not permitted as defined in *4.4.6 Import/Export semantics* in ISO/IEC 19775-1. For example, adding the imported node as a child to another node in the current scene.

5.3.6 SAI_INSUFFICIENT_CAPABILITIES

The user is attempting to add a node to an execution context that is greater than the capabilities defined by the profile and components definition for that scene.

5.3.7 SAI_INVALID_ACCESS_TYPE

The attempt to perform an operation of a field failed because it is an invalid action for that field type. For example, an attempt made to read the value of an inputOnly field would generate this error.

5.3.8 SAI_INVALID_BROWSER

The instance of SAIBrowserRef data type provided as part of the parameters to the service request has been disposed of prior to this request.

5.3.9 SAI_INVALID_DOCUMENT

When the user has attempted to import a World Wide Web Consortium Document Object Model (DOM) document into an X3D scene and the document cannot be completely resolved to an X3D scene graph. There are many cases where this error might be generated, including the following:

Example 1 an invalid document structure

Example 2 not having the correct root element

Example 3. No external HTML pag

5.3.10 SAI_INVALID_EXECUTION_CONTEXT

The instance of SAIExecutionContext data type provided as part of the parameters to this service request has been disposed of prior to this request.

5.3.11 SAI_INVALID_FIELD

The instance of SAIField data type provided as part of the parameters to this service request has been disposed of prior to this request.

5.3.12 SAI_INVALID_NAME

The name provided to a service request is invalid or cannot be found in the context of that object.

5.3.13 SAI_INVALID_NODE

The instance of SAINodeID data type provided as part of the parameters to this service request has been disposed of prior to this request.

5.3.14 SAI_INVALID_OPERATION_TIMING

The user is attempting to make a service request that is performed outside of the context within which such operations are permitted (see 4.8.3.7 User code lifecycle). Where a service defines this as being a possible error type, this shall only be thrown by internal interactions. External interactions shall never generate this error.

5.3.15 SAI_INVALID_URL

An instance of SAIURL data type provided in one or more of the parameters to this service request are invalid due to a syntax error. Errors due to the requested URL not being available shall generate either an SAI_URL_UNAVAILABLE error or an asynchronous event notifying of such a problem.

5.3.16 SAI_INVALID_X3D

The SAIStream, SAIString or X3D file (for example, as a result of the fetching of a URL reference) passed to this service request contains invalid syntax and cannot be parsed to produce legal data types for use in other service requests.

5.3.17 SAI_NODE_IN_USE

Indication that a named node handling action has attempted to re-use a name that is already defined elsewhere in this current scene.

EXAMPLE A user is attempting to import a node as a name that is already described by a DEF.

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

5.3.18 SAI_NODE_NOT_AVAILABLE

field connections

An error condition used for IMPORTed nodes. The user has described a node that the IMPORT statement has said is valid, but the underlying Inline has not yet been loaded to verify that it is a correctly EXPORTed node.

awk

5.3.19 SAI_NOT_SHARED

A service request was made that assumed the browser was currently participating in a shared scene graph when it was not.

5.3.20 SAI_NOT_SUPPORTED

Generalised error for when a service request is made for a capability that is not available in this browser implementation. For example, if the user requests a profile declaration for a profile that is not supported by the browser, this error may be generated.

5.3.21 SAI_URL_UNAVAILABLE

The service request requiring the browser to have a world URL set cannot be completed because no URL has been set. This error is typically generated from a `getWorldURL` (6.4.6 getWorldURL) or `getNode` (6.4.7 getNode) service request.

5.4 Event types

5.4.1 Overview

Browser event types are asynchronous events that are generated in response to changes in the status of the browser implementation. The following event types shall be implemented by each language binding. Additional implementation dependent events may be defined to supplement the provided event types.

5.4.1.1 SAI_Browser_Connection_Error

The event type representing an error condition has occurred in the internal connection between the browser and the external application (see 4.5.3.5 Connection Lost).

5.4.1.2 SAI_Browser_Event

The event type that represents the general class of events produced by each browser service (see 4.5.3 Browser to external application).

5.4.1.3 SAI_Browser_Initialized

The event type representing the browser having completed the initialisation process of loading X3D content (see 4.5.3.2 Initialize).

5.4.1.4 SAI_Browser_Shutdown

The event type representing the browser being shutdown. That is the execution model is no longer running or content is displayed (see 4.5.3.3 Shutdown).

5.4.1.5 SAI_Browser_URL_Error

The event type representing an error condition when no URLs could be processed to form valid X3D content (see 4.5.3.4 No URLs Available).

5.4.2 SAIFieldEvent

The event type used to represent the notification of a change in a field value that the external application has registered interest in.

Extensible 3D (X3D)
Part 2: Scene access interface (SAI)

6 Services reference

6.1 Introduction and topics

6.1.1 Introduction

This clause provides a detailed definition of the semantics of the services that a browser shall provide to external applications as defined in this part of ISO/IEC 19775.

6.1.2 Topics

Table 6.1 specifies the topics for this clause.

Table 6.1 — Topics

getStatement
createStatement

getMeta ?
setMeta ?

X3D version (Execution Context in Scene : getSpecificationVersion
for browser: getBrowser Properties

6.1.3 Conventions used

Each of the services in this clause defines a particular request that can be made through the SAI. Each request is defined by a number of characteristics. In Table 6.2 the first column defines each characteristic type and the second defines the properties of that characteristic.

Table 6.2 — SAI request conventions

parameters:	first param data type, second param, [optional parameter data type], [multiple optional parameter data types]
returns:	The list of return value data types or expected ranges
errors:	List of errors data types
events:	The first event The second event
buffered:	Simple yes, no or N/A
external:	Yes if this is available only to an external interface, No if it is available to both internal and external interfaces.

Parameters are listed by data type and are shown separated by a comma (,) and a space. A parameter shown in square brackets [] indicates a single optional value of the data type specified within the brackets. The "[]s" symbology, square brackets followed by the "s" character, indicates multiple optional parameters of that type are allowed. For example, [SAIURL]s indicates that multiple instances of the data type SAIURL may be provided while [SAIURL] indicates that only a single SAIURL instance may be provided.

All characteristics defined for every service shall be implemented for each language binding. At the end of each table, explanatory text includes ~~extra~~ additional information pertinent to the implementation of that service.

6.2 Establishing a connection

6.2.1 Introduction

The following services can be used to establish a session and obtain a browser reference. Individual browser implementations may support one or both of these methods. At least one service shall be supported.

6.2.2 getBrowser

parameters:	SAIParameterList
returns:	SAIBrowserRef
errors:	SAI_BROWSER_UNAVAILABLE
events:	none
buffered:	N/A

The `getBrowser` service returns a reference to an instance of an X3D browser through which other service requests may be processed.

This is a blocking call. No further requests from this external application will be processed until an SAIBrowser value has been generated (which may include the need to start a new instance of an X3D browser) or an error condition is generated.

If an application makes a request for the same browser twice in the same session then the same browser identifier shall be returned.

An implementation may define more than one variant of this service with different parameter types. For example there may be alternate forms to access a browser embedded in a HTML page and one for remote access from another machine within the same language binding.

Additional error types may be added by individual language bindings to deal with platform specific issues.

6.2.3 createBrowser

parameters:	SAIParameterList, SAIPropertyList
returns:	SAIBrowserApp
errors:	SAI_BROWSER_UNAVAILABLE

events:	none
buffered:	N/A

The `createBrowser` service creates a new instance of a browser application. The browser shall start with no URL set (that is, no active X3D scene graph). The URL may be set at a later time using the `loadURL` (see 6.3.14 loadURL) or `replaceWorld` (see 6.3.12 replaceWorld) service requests.

The property list is used to define the properties of the browser application itself. The service request shall use the same property list definitions as those defined in `loadURL` (see 6.3.14 loadURL). It is not required to support exactly the same capabilities, but the property list format shall be identical and any behaviours are identical.

This is a blocking request. No further requests from this external application will be processed until a new instance of an X3D browser has been created or an error condition is generated.

Each request of this service shall produce a new browser application instance in accordance with the supplied parameter values.

An implementation may define more than one variant of this service with different parameter types. For example there may be alternate forms to start a browser on a remote machine or to create a new window within a running application.

Additional error types may be added by individual language bindings to deal with platform specific issues.

Individual language bindings may add extra calls to the `SAIBrowserApp` to provide platform-specific low-level handles for the language.

EXAMPLE A language binding may allow access to the raw image pixel data for an offline image renderer so that a user may use platform-specific calls to make extra drawing and compositing actions.

6.3 Browser services

6.3.1 Introduction

The following services can be requested from a browser. Although not specified repeatedly, all services are capable of throwing an SAI_CONNECTION_ERROR whenever a request is made if the session between the application and the browser has failed.

NOTE The data representation of the parameters or return values is not specified. It is equally valid to represent all parameters as strings as it is for binary representations.

6.3.2 getName

parameters:	SAIBrowserRef
returns:	SAIBrowserName
errors:	SAI_DISPOSED
events:	None
buffered:	

	No
external:	No

The getName service returns the name of the browser. This name is implementation dependent. If this service is not supported a NULL value shall be returned.

6.3.3 getVersion

parameters:	SAIBrowserRef
returns:	SAIBrowserVersion
errors:	SAI_DISPOSED
events:	None
buffered:	No
external:	No

The getVersion service returns the current version of the browser application. The version number of the browser is implementation dependent. If this service is not supported then a NULL value shall be returned.

6.3.4 getCurrentSpeed

parameters:	SAIBrowserRef, SAILayerID
returns:	SAINavSpeed
errors:	SAI_DISPOSED
	SAI_INVALID_OPERATION_TIMING
events:	None
buffered:	No
external:	No

The getCurrentSpeed service returns the navigation speed of the current world. The current speed is the average navigation speed for the currently bound NavigationInfo node of the active layer in base speed units in the coordinate system of the currently bound viewpoint.

6.3.5 getCurrentFrameRate

parameters:	SAIBrowserRef
returns:	SAIFrameRate
errors:	SAI_DISPOSED
	SAI_INVALID_OPERATION_TIMING
events:	None
buffered:	No
external:	No

The getCurrentFrameRate service returns the current frame display rate of the browser in frames per second. If this is not supported, the value returned is zero.

capability

6.3.6 getSupportedProfiles

parameters:	SAIBrowserRef
returns:	SAIProfileDeclaration [SAIProfileDeclaration]s
errors:	SAI_DISPOSED
events:	None
buffered:	

	No
external:	No

The getSupportedProfiles service returns the list of all profiles that are supported by this browser. All browsers shall support at least one profile. It shall be an error if the browser returns a declaration for a profile that it does not fully support.

6.3.7 getProfile

parameters:	SAIBrowserRef, SAIString
returns:	SAIProfileDeclaration
errors:	SAI_DISPOSED
	SAI_NOT_SUPPORTED
events:	None
buffered:	No
external:	No

The getProfile service returns the declaration of the named profile. The value of the SAIString parameter is the name of a profile from which to fetch the declaration and shall conform exactly to the name specified in ISO/IEC 19775-1. It shall be an error if a name with the wrong case, incorrect spelling, or anything other than an exact match is provided. The browser is only required to return an SAIProfileDeclaration value if it supports the named profile. If it does not support the named profile, SAI_NOT_SUPPORTED shall be generated.

6.3.8 getSupportedComponents

parameters:	SAIBrowserRef
returns:	SAIComponentDeclaration [SAIComponentDeclaration]s
errors:	SAI_DISPOSED
events:	None
buffered:	No
external:	No

[Duplicate?]

The getSupportedComponents service returns a list of all components that are supported by this browser. All browsers shall support at least one component, as required to support profiles.

6.3.9 getComponent

parameters:	SAIBrowserRef, SAIComponent
returns:	SAIComponentDeclaration
errors:	SAI_DISPOSED
	SAI_NOT_SUPPORTED
events:	None
buffered:	No
external:	No

The getComponent service returns the declaration of the named component. The value of the SAIComponent parameter is the name of a component and level from which to fetch the declaration and shall conform exactly to the naming conventions used in the file format. It shall be an error if the user provides a name with the wrong case, incorrect spelling or anything other than an exact

match. The browser is only required to return a SAIComponentDeclaration value if it supports the named component and the requested level. If it does not support the component at the level desired, SAI_NOT_SUPPORTED shall be generated.

6.3.10 getExecutionContext

parameters:	SAIBrowserRef
returns:	SAIExecutionContext
errors:	SAI_DISPOSED
	SAI_INVALID_OPERATION_TIMING
events:	None
buffered:	No
external:	No

The `getExecutionContext` service returns the current execution context. If used in an internal interaction, this service returns the execution context in which the containing node exists (see 4.4.3 Containing Node). When used in an external interaction, this service returns the current top-level scene.

The execution context is the base form of a scene, but only provides access to the local nodes, PROTOs and routes as defined by the X3D name scoping rules as defined in *4.4.7 Run-time name scope* in ISO/IEC 19775-1. Depending on the place in the scene graph, the returned type may be an instance of SAIScene allowing the user to utilize the greater capabilities.

6.3.11 createScene

parameters:	SAIBrowserRef, [SAIProfileDeclaration], [SAIComponentDeclaration]s
returns:	SAIScene
errors:	SAI_DISPOSED
	SAI_INVALID_OPERATION_TIMING
events:	None
buffered:	No
external:	No

The `createScene` service creates a new empty scene that conforms to the given profile and component declarations. Although the specification does not require either be provided, it shall be an error to provide neither profile nor component definitions. A user shall provide at least one valid profile or component identifier to this request.

A scene created this way shall always have its specification version set to "3.0", "3.1", "3.2", or "3.3" (as appropriate) and the encoding set to "Scripted".

or "4.0"

2.

6.3.12 replaceWorld

parameters:	SAIBrowserRef, SAIScene
returns:	None
errors:	SAI_INVALID_SCENE
	SAI_DISPOSED
	SAI_INVALID_OPERATION_TIMING
events:	SAI_Browser_Shutdown
	SAI_Browser_Initialized
buffered:	

	No
external:	No

The replaceWorld service replaces the current world with the world specified by the SAIScene parameter. If another replaceWorld or loadURL (see 6.3.14 loadURL) request is made during the processing of the current request, the current request is terminated and the new one started. In this case, no extra shutdown event shall be generated. The initialize event shall be generated at the point where the world is ready to be run. The scene is not required to contain any valid content. Setting a value of NULL shall clear the currently set scene and leave a blank browser with no renderable content and no current scene.

The SAI_Browser_Shutdown event is generated immediately upon receiving this service request.

The SAI_Browser_Initialized event is generated when the new nodes have been set and all browser specific initialization has taken place but before the first time driven event cascade has been started (event cascades may have previously resulted due to the initialization process through internal scripts).

6.3.13 importDocument

parameters:	SAIBrowserRef, DOMNode
returns:	SAIScene
errors:	SAI_INVALID_DOCUMENT SAI_DISPOSED SAI_INVALID_OPERATION_TIMING SAI_NOT_SUPPORTED
events:	None
buffered:	No
external:	No

The importDocument service is a utility request to import a World Wide Web Consortium (W3C) Document Object Model (DOM) document or document fragment and convert it to an X3D scene. The input allows any form of DOM Node as defined by 2.[W3CDOM2]. Although all derived types are available, it shall only be required that DOCUMENT, DOCUMENT_FRAGMENT, and ELEMENT types are required to be supported for the conversion process. The method only performs a conversion process and does not display the resulting scene. The scene may then be used as the argument for the replaceWorld (see 6.3.12 replaceWorld) service. When the conversion is made, there is no lasting connection between the DOM and the generated scene. Each request shall be a single conversion attempt (the conversion may not be successful if the DOM does not match the X3D scene graph structure).

Support for this method is optional and shall be dependent on the browser support for the XML encoding (see ISO/IEC 19776-1). If the browser implementation supports the XML encoding, it shall support this service. If the browser does not support the XML encoding, the implementation may support this service. User code may check that this service is supported through the checking the browser properties with the getBrowserProperties service. If this service is not supported by the browser implementation, SAI_NOT_SUPPORTED error shall be generated.

6.3.14 loadURL

parameters:	SAIBrowserRef, SAIURL [SAIURL]s, SAIPropertyList
returns:	None
errors:	SAI_INVALID_URL SAI_INVALID_OPERATION_TIMING SAI_DISPOSED
events:	SAI_Browser_Shutdown SAI_Browser_Initialize SAI_Browser_URL_Error
buffered:	No
external:	No

The `loadURL` service inserts the content identified by the URL(s) in the current world under control of the contents of the SAIPropertyList instance.

The SAI_Browser_Shutdown event is generated immediately upon receiving this service request if the parameter list is such that the browser is about to be shutdown (EXAMPLE replaces an HTML Frame where the browser was embedded).

The SAI_Browser_Initialized event is generated when the new nodes have been set and all browser specific initialization has taken place but before the first time driven event cascade has been started (event cascades may have previously resulted due to the initialization process through internal scripts).

The property list definition shall include at least one property that defines loading the URL supplied as a new world in the supplied SAIBrowserRef. If the property list is empty, the action is to replace the world of the current browser with the new world if the successful URL is an X3D file.

If another `replaceWorld` (see 6.3.12 replaceWorld) or `loadURL` request is made during the processing of the current request, the current request is terminated and the new one started. In this case, no extra shutdown event shall be generated. The SAI_Browser_Initialize event shall be generated at the point where the world is ready to be run if `replaceWorld` was called.

6.3.15 setDescription

parameters:	SAIBrowserRef, SAIString
returns:	None
errors:	SAI_INVALID_OPERATION_TIMING SAI_DISPOSED
events:	None
buffered:	No
external:	No

If the browser supports a description title, it shall be set to the new description. Typically, this will be the title in a window title bar. In cases where there may be multiple browsers on a single window, the result is implementation dependent.

6.3.16 createX3DFromString

This functionality can override the provided WorldInfo title in the scene.

parameters:	SAIBrowserRef, SAIString

returns:	SAIScene
errors:	SAI_INVALID_X3D
	SAI_INVALID_OPERATION_TIMING
	SAI_DISPOSED
	SAI_NOT_SUPPORTED
events:	None
buffered:	No
external:	No

The `createX3DFromString` service creates nodes from a string. The string shall contain valid X3D syntax; otherwise an error is generated. If any relative URLs are encountered in this string, the base is assumed to be the current browser location. The string is not required to contain the X3D file header. If it is present, it shall be treated as an indicator to the version of X3D contained. If absent, the default version assumed shall be that specified in *7.2.5.2 Header statement* in ISO/IEC 19775-1. A browser is not required to support any versions prior to ISO/IEC 19775.

If the string contains legal X3D statements but does not contain any node instances, a valid SAIScene value shall still be returned containing no root nodes, but with the appropriate declaration identifiers. For example the string may contain EXTERNPROTO declarations but no instances of any node. If the SAIString provides the content in an encoding format that the browser implementation does not support, the browser shall generate an SAI_NOT_SUPPORTED error.

6.3.17 createX3DFromStream

parameters:	SAIBrowserRef, SAIStream
returns:	SAIScene
errors:	SAI_INVALID_X3D
	SAI_INVALID_OPERATION_TIMING
	SAI_DISPOSED
	SAI_NOT_SUPPORTED
events:	None
buffered:	No
external:	No

The `createX3DFromStream` service creates nodes from an arbitrary, user-provided stream of input data. The stream shall contain valid X3D syntax from the first character; otherwise, an error is generated. If any relative URLs are encountered in this string, the base is assumed to be the current browser location. The stream is required to include the X3D File Header in accordance with the encoding requirements for the format.

If the string contains legal X3D statements but does not contain any node instances, a valid SAIScene value shall still be returned containing no root nodes, but with the appropriate declaration identifiers. For example, the string may contain EXTERNPROTO declarations but no instances of any node. If the stream identified by SAIStream provides the content in an encoding format that the browser implementation does not support, the browser shall generate an SAI_NOT_SUPPORTED error.

6.3.18 createX3DFromURL

parameters:	SAIBrowserRef, SAIURL [SAIURL]s
returns:	SAIScene
errors:	SAI_INVALID_URL SAI_INVALID_OPERATION_TIMING SAI_DISPOSED
events:	SAI_Browser_URL_Error
buffered:	No
external:	No

The createX3DFromURL service creates nodes from the contents of the file represented by the URL. The URL may be a relative URL which is considered to be using the browser location as the base document. The scene described by that URL shall be identified by the returned SAIScene value.

6.3.19 updateControl

parameters:	SAIBrowserRef, SAIAction
returns:	None
errors:	SAI_DISPOSED
events:	None
buffered:	N/A
external:	Yes

The updateControl specifies the manner in which buffered updates are processed.

The SAIAction parameter specifies the actions that may be applied against the buffer. Other actions may be added, such as to query the number of items, or the state of the buffer and are implementation dependent. Table 6.3 defines the actions specified in this part of ISO/IEC 19775.

Table 6.3 — updateControl SAIAction values

Service	Action Type
updateControl	BeginUpdate
	EndUpdate

The timestamp of events generated at the call to EndUpdate are implementation dependent but should be consistent with the time within the current world. That is, timestamps cannot be in the "past" relative to the other current events generated internally with event model at the time when they are generated.

BeginUpdate and EndUpdate are not nesting calls. Once BeginUpdate has been called, it may be called any number of times, but only a single EndUpdate call is needed to release the buffered events into the scene graph. A call to EndUpdate without a previous matching BeginUpdate has no effect.

6.3.20 registerBrowserInterest

parameters:	SAIBrowserRef, SAIAction, SAIRequester
returns:	None
errors:	SAI_INVALID_OPERATION_TIMING

	SAI_DISPOSED
events:	Receiver of all SAIBrowserEvents
buffered:	No
external:	Yes

The registerBrowserInterest service nominates the requester as the receiver of all SAIBrowserEvents. The act of making this service request itself does not imply any events shall be generated.

The parameter of type SAIRequester could be inferred from the source of the input and may not need to be part of the parameters and is implementation dependent. Each binding to this service shall specify this requirement.

The parameter of type SAIAction specifies whether this is a request to add interest in events, or to remove interest in the events. Table 6.4 defines the actions specified in this part of ISO/IEC 19775.

Table 6.4 — registerBrowserInterest SAIAction values

Service	Action Type
registerBrowserInterest	AddBrowserInterest
	RemoveBrowserInterest

Any change to the current browser shall be sent to the listener that has registered interest in these events. These event notifications shall be made independent of the method of communicating with the browser.

As a minimum, a conforming implementation shall provide the events defined in 4.5.3 Browser to External Application.

6.3.21 getRenderingProperties

parameters:	SAIBrowserRef
returns:	SAIPropertyList
errors:	SAI_INVALID_OPERATION_TIMING
	SAI_DISPOSED
events:	None
buffered:	No
external:	No

The getRenderingProperties service is used to query for the rendering capabilities of the browser. This gives a list of the low-level hardware capabilities of the browser rather than what X3D components are supported. For example, it will give the user an idea of how many multitexture units can be handled and allows the end user to customize the number of nodes to use in the MultiTexture node. The keys and values in the property list are implementation dependent and are for informative purposes only. Table 6.5 lists the properties that are returned by this service.

Table 6.5 — Standard rendering property definitions

Property Name	Value data type	Description
Shading	String	The type of shading algorithm in use. Typical values are Flat, Gouraud, Phong, Wireframe.
MaxTextureSize	String	The maximum texture size supported. The format shall be WIDTHxHEIGHT describing the number of pixels in each direction (for example 1024x1024).
TextureUnits	Integer	The number of texture units supported for doing multitexture.
AntiAliased	Boolean	True or false if the rendering is currently anti-aliased or not
ColorDepth	Integer	The number of bits of colour depth supported by the screen. Allows for optimized selection of textures, particularly for lower colour depth screen capabilities.
TextureMemory	Float	The amount of memory in megabytes available for textures to be placed on the video card.

The user shall not be able to directly effect the rendering properties of the browser by modifying the properties returned by this service.

6.3.22 getBrowserProperties

parameters:	SAIBrowserRef
returns:	SAIPropertyList
errors:	SAI_INVALID_OPERATION_TIMING SAI_DISPOSED
events:	None
buffered:	No
external:	No

The `getBrowserProperties` service is used to query for the capabilities of the browser reference itself. This gives a list of the expanded interfaces that this browser reference is capable of supporting. For example it may be used to query for the existence of browser implementation-specific extensions to the defined browser class or future extensions as provided by this specification.

Table 6.6 defines some standard property names that are reserved by part of ISO/IEC 19775. Where a browser implementer chooses to add additional capabilities, the naming convention of these additional properties shall follow the guidelines defined in *4.1.3 Conventions* used in part 1 of of ISO/IEC 19775 (see 2.[I19775-1]).

Table 6.6 — Standard properties describing extension capabilities

Property Name	Value	Description

	data type	
ABSTRACT_NODES	Boolean	The browser implementation supports the ability to describe each node type with interfaces that correspond to the abstract node types as defined in ISO/IEC 19775-1 in addition to the basic requirement to support the X3DNode abstract type. This indicates that the browser supports at least Conformance Level 2.
CONCRETE_NODES	Boolean	The browser implementation supports the ability to describe each node type with interfaces that correspond to the concrete node types as defined in ISO/IEC 19775-1 in addition to the requirement to support all of the abstract types. This indicates that the browser supports at least Conformance Level 3.
EXTERNAL_INTERACTIONS	Boolean	This SAIBrowserRef supports the additional services required by external interfaces. A SAIBrowserRef provided to user code in internal interactions shall not set this property.
PROTOTYPE_CREATE	Boolean	The browser implementation supports the ability to dynamically create PROTO and EXTERNPROTO representations through service requests. The basic service capability only allows the ability to create instances of predefined PROTO structures read from a file format.
DOM_IMPORT	Boolean	The browser implementation supports the importDocument service request.
XML_ENCODING	Boolean	The browser supports XML as a file format encoding.

CLASSIC_VRML_ENCODING	Boolean	The browser supports the Classic VRML encoding.
COMPRESSED_BINARY_ENCODING	Boolean	The browser supports the binary file format encoding.

6.3.23 changeViewpoint

parameters:	SAIBrowserRef, SAIAction, SAILayerID
returns:	None
errors:	SAI_INVALID_OPERATION_TIMING SAI_DISPOSED
events:	None
buffered:	Yes
external:	No

The `changeViewpoint` service changes the currently bound *X3DViewpointNode* instance on the specified Layer to the instance defined by the action. Valid action types are previous, next, first and last. If a layer ID is not specified, the current activeLayer is used. When the viewpoint is changed using this service request, the browser shall first unbind the current instance and then bind the new instance. That is, the number of items on the bindable stack bindable nodes shall not increase as a result of making this service request. Table 6.7 defines the actions specified in this part of ISO/IEC 19775.

Table 6.7 — changeViewpoint SAIAction values

Service	Action Type
changeViewpoint	Next
	Previous
	First
	Last

This service request implies that there is a standard, well-known ordering of the *X3DViewpointNode* instances so that consistent visual behaviour shall be observed. The order shall be based on the following rules:

1. The order is declared in the originally parsed file or stream, including resolution of PROTO instances, but not including EXTERNPROTO or *X3DInlineNode* instances.
2. Dynamically created node instances are always appended.
3. Instances located in *X3DInlineNode* instances and EXTERNPROTO instances shall be in the order in which the external scene is resolved, and appended to the list. The inclusion of these external instances is also dependent on the browser property EnableInlineViewpoints.

An invalid SAILayerID shall result in the operation being ignored. Requests for SAILayerID values less than zero or greater than or equal to the number of defined layers are considered invalid and shall cause error SAI_INVALID_OPERATION_TIMING to be issued.

If the world only contains the default *X3DViewpointNode* instance, this request has no effect on the visual output.

6.3.24 print/println

parameters:	SAIBrowserRef, SAIString
returns:	None
errors:	None
events:	SAI_DISPOSED
buffered:	No
external:	No

The `print` service prints a message to the browser's console. The language-specific bindings may provide overloaded variations on this service that do not take an SAIString value, but take other data types. Other variants may include the ability to automatically add linefeed/newline characters without the need to explicitly declare them in the SAIString value. A binding shall provide at least the base SAIString variant (print) and a variant that appends linefeed/newline characters (println).

User code may call this service at any time, without restriction, unless the browser reference has been disposed of.

6.3.25 dispose

parameters:	SAIBrowserRef
returns:	None
errors:	SAI_INVALID_OPERATION_TIMING
events:	SAI_B_Shutdown
buffered:	No
external:	Yes

The `dispose` service indicates that the client is about to exit this session and the browser is free to dispose of any resources that this client may have consumed. An SAI_Browser_Shutdown event is sent only to this client and may be generated internally by the language implementation on the client machine (that is, it is not required that the browser itself generate this event, just that the event is generated). If any events have been held because `BeginUpdate` has been called, disposing of the browser shall also call `EndUpdate` to release those events to the browser for final processing.

6.3.26 setBrowserOption

parameters:	SAIBrowserRef, SAIString, SAIObject
returns:	SAIBoolean
errors:	SAI_INVALID_OPERATION_TIMING
events:	None
buffered:	No

external:	Yes

The `setBrowserOption` service allows setting options defined in *9.2.4 Browser options* in ISO/IEC 19775-1. The name field shall be one of the defined names in Table 9.2 in ISO/IEC 19775-1. This service shall return an SAIBoolean value indicating whether the change request was successful. A browser is not required to support dynamic changes to any options. If a browser option is not supported, a value of FALSE shall be returned.

6.4 Execution context services

6.4.1 getSpecificationVersion

parameters:	SAIExecutionContext
returns:	SAIString
errors:	SAI_INVALID_OPERATION_TIMING
	SAI_DISPOSED
events:	None
buffered:	No
external:	No

The getSpecificationVersion returns the version string that describes to which specification version this scene adheres. This version represents the appropriate version number as defined in ISO/IEC 14772-1, ISO/IEC 19775-1, or has value 1.0 for versions of VRML that precede the specification in ISO/IEC 14772-1 that are supported by the implementation.

6.4.2 getEncoding

parameters:	SAIExecutionContext
returns:	SAIEncoding
errors:	SAI_INVALID_OPERATION_TIMING
	SAI_DISPOSED
events:	None
buffered:	No
external:	No

The `getEncoding` service returns the encoding type that was used to produce the portion of the scene represented by the specified execution context. The encoding is one of a fixed set, but may include additional types that are browser implementation specific. The minimum required set of values (but not necessarily supported by the browser implementation) shall be:

1. *Scripted*: For scenes that are created completely through the SAI and did not originate through a file somewhere.
2. *ASCII*: For VRML 1.0 specification files.
3. *VRML*: For VRML and the X3D Classic VRML encoding specified in ISO/IEC 19776-2.
4. *XML*: For X3D XML-encoded files specified in ISO/IEC 19776-1.
5. *CompressedBinary*: for X3D Compressed binary-encoded files specified in ISO/IEC 19776-3.

6. *BIFS*: For MPEG-4 BIFS-encoded format specified in ISO/IEC 14496-1.

6.4.3 getProfile

parameters:	SAIExecutionContext
returns:	SAIProfileDeclaration
errors:	SAI_INVALID_OPERATION_TIMING
	SAI_DISPOSED
events:	None
buffered:	No
external:	No

The `getProfile` service returns the profile that is used to describe this scene. If the specification version is for a specification version prior to X3D, the profile shall be VRML. If no profile is provided, this shall return `NULL`.

6.4.4 getComponents

parameters:	SAIExecutionContext
returns:	[SAIComponentDeclaration]s
errors:	SAI_INVALID_OPERATION_TIMING
	SAI_DISPOSED
events:	None
buffered:	No
external:	No

The `getComponents` service returns the component(s) used to describe the scene. The list returned shall represent only explicit component declarations and not the implied components from the profile declaration. If no component definitions are set, `NULL` shall be returned.

6.4.5 getUnits

parameters:	SAIExecutionContext
returns:	[SAIUnitDeclaration]s
errors:	SAI_INVALID_OPERATION_TIMING
	SAI_DISPOSED
events:	None
buffered:	no
external:	no

The `getUnits` service returns all of the units used to describe the scene. The list returned shall represent all explicit unit declarations and the currently applied default units.

6.4.6 getWorldURL

parameters:	SAIExecutionContext
returns:	SAIURL
errors:	SAI_INVALID_OPERATION_TIMING
	SAI_DISPOSED
events:	None
buffered:	no
external:	no

The `getWorldURL` service returns the fully qualified URL of this scene. This returns the entire URL including any possible arguments that might be associated with a CGI call or similar mechanism. If the world was created entirely programmatically, the URL shall be `NULL`.

6.4.7 getNode

TODO getStream

parameters:	SAIExecutionContext, SAIString, SAIAction
returns:	SAINode
errors:	SAI_INVALID_OPERATION_TIMING SAI_INVALID_NAME SAI_DISPOSED SAI_NODE_NOT_AVAILABLE
events:	None
buffered:	No
external:	No

The `getNode` service searches for a node based on specified criteria and returns an identifier for the node.

The SAIString is to identify the name of the node that has been marked with one of the naming statements DEF, IMPORT or EXPORT in the currently loaded X3D scene or previously added with a `namedNodeHandling` request (see 6.4.10 namedNodeHandling).

The SAIAction shall indicate which of the naming types shall be used to find the node. For example, providing an action of `ImportNode` shall not return a name that may be valid, but describes a node named with the DEF statement. Table 6.8 defines the actions specified in this part of ISO/IEC 19775.

Table 6.8 — getNode SAIAction values

Service	Action Type
getNode	DEFNode
	IMPORTNode
	EXPORTNode

Access shall only be available to names in this scene. DEFs in Inlined files shall not be accessible in accordance with *4.4.3 DEF/USE Semantics* and *4.4.6, Import/Export semantics* in ISO/IEC 19775-1.

If the SAIAction is `IMPORTNode` and the name is valid but the node definition is not yet available from the source Inline node, SAI_NODE_NOT_AVAILABLE shall be generated.

6.4.8 createNode

TODO Create Stream

parameters:	SAIExecutionContext, SAIString
returns:	SAINode
errors:	SAI_INVALID_OPERATION_TIMING

	SAI_DISPOSED
	SAI_INVALID_NAME
events:	None
buffered:	No
external:	No

The createNode service creates a new default instance of the node given by the SAIString value containing the name of an X3D node type. The availability of the node is defined by the containing scene's profile and component declarations. The name shall only refer to a built-in node and shall not be used to create instances of PROTOs or EXTERNPROTOs. If the node is not available in the currently specified profile and components, the browser shall issue the SAI_INVALID_NAME error.

6.4.9 createProto

parameters:	SAIExecutionContext, SAIString
returns:	SAINode
errors:	SAI_INVALID_OPERATION_TIMING
	SAI_DISPOSED
	SAI_INVALID_NAME
events:	None
buffered:	No
external:	No

The createProto service creates a new default instance of the named PROTO. The naming and scoping rules for creating a proto instance for which the current execution context is inside another proto are defined by *4.4.7 Run-time name scope* in ISO/IEC 19775-1. If there is no PROTO declaration available that matches the given name, the browser shall generated the SAI_INVALID_NAME error.

6.4.10 namedNodeHandling

parameters:	SAIExecutionContext, SAIAction, SAIAction, SAIString,
	[SAINode | SAIString, [SAIString]]
returns:	None
errors:	SAI_INVALID_OPERATION_TIMING
	SAI_DISPOSED
	SAI_IMPORTED_NODE
	SAI_NODE_IN_USE
	SAI_INVALID_NAME
events:	None
buffered:	Yes
external:	No

The namedNodeHandling service is a request to add, remove, or update the node identified by the SAIString value where that name is considered to use the DEF, or IMPORT semantics. The add/remove/update shall be described by the first SAIAction value. If the name already exists as a mapping, the current mapping is replaced with the new mapping. When adding a new named node, the new named node is not required to be part of this scene.

The second SAIAction value describes which of the DEF or IMPORT naming facilities shall be the target of this service request. This ensures that correct semantics are applied. If the action is to add and the name is already registered, SAI_NODE_IN_USE is generated. If the action is to replace or update, and the node is not already registered, the implementation may treat this as an add request. Table 6.9 defines the actions specified in this part of ISO/IEC 19775.

Table 6.9 — namedNodeHandling SAIAction values

Service	Action Type
namedNodeHandling	AddDEFNode/UpdateDEFNode
	RemoveDEFNode
	AddIMPORTNode/UpdateIMPORTNode
	RemoveIMPORTNode
	AddEXPORTNode/UpdateEXPORTNode
	RemoveEXPORTNode

The first SAIString value identifies a name with a node as it should be known in this scene. The name is not an intrinsic property of the node and this only serves as a mapping function.

The second argument provides an option depending on the action being undertaken. SAINode value is a reference to the node that may be needed for verification of the DEF name addition. For adding IMPORTs, the second string shall be the exported node name in the DEF'd inline and the optional third string shall be the name to store it as in this scene.

6.4.11 getProtoDeclaration

parameters:	SAIExecutionContext, SAIString
returns:	SAIProtoDeclaration
errors:	SAI_INVALID_OPERATION_TIMING SAI_INVALID_NAME SAI_DISPOSED
events:	None
buffered:	No
external:	No

The `getProtoDeclaration` service returns the named PROTO declaration representation from this scene. This shall only be used to request PROTO declarations. A request for an EXTERNPROTO declaration shall generate SAI_INVALID_NAME.

6.4.12 protoDeclarationHandling

parameters:	SAIExecutionContext, SAIString, SAINode, SAIAction
returns:	None
errors:	

SAI_INVALID_OPERATION_TIMING
SAI_DISPOSED

events: None
buffered: Yes
external: No

The protoDeclarationHandling service is a request to add, remove or change the ProtoDeclaration identified by the SAIString value.

The SAIAction parameter specifies whether the service request is an add or removal of the declaration node. If the name already exists as a mapping, the current mapping is replaced with the new map. When adding a new declaration it may come from another scene. Table 6.10 defines the actions specified in this part of ISO/IEC 19775.

Table 6.10 — protoDeclarationHandling SAIAction values

Service	Action Type
protoDeclarationHandling	AddProto/UpdateProto
	RemoveProto

6.4.13 getExternProtoDeclaration

parameters: SAIExecutionContext, SAIString
returns: SAIProtoDeclaration
errors: SAI_INVALID_OPERATION_TIMING
SAI_INVALID_NAME
SAI_URL_UNAVAILABLE
SAI_DISPOSED
events: None
buffered: No
external: No

The getExternProtoDeclaration service returns the named EXTERNPROTO declaration representation from this scene. This shall only be used to request an EXTERNPROTO declaration. A request for a PROTO declaration shall generate SAI_INVALID_NAME.

6.4.14 externprotoDeclarationHandling

parameters: SAIExecutionContext, SAIString, SAINode, SAIAction
returns: None
errors: SAI_INVALID_OPERATION_TIMING
SAI_DISPOSED
events: None
buffered: Yes
external: No

The externprotoDeclarationHandling service is a request to add, remove or update the ExternProtoDeclaration identified by the SAIString value.

The SAIAction parameter is used to indicate if the service request is an add or removal of the declaration node. If the name already exists as a mapping, the current mapping is replaced with the new map. When adding a new declaration, it may come from another scene. Table 6.11 defines the actions specified in this part of ISO/IEC 19775.

Table 6.11 — externprotoDeclarationHandling SAIAction values

Service	Action Type
externprotoDeclarationHandling	AddExternProto/UpdateExternProto
	RemoveExternProto

6.4.15 getRootNodes

parameters:	SAIExecutionContext
returns:	SAINodes
errors:	SAI_INVALID_OPERATION_TIMING
	SAI_INVALID_NAME
	SAI_DISPOSED
events:	None
buffered:	No
external:	No

The getRootNodes service returns a listing of the current root nodes of the execution context. If the context was generated from a file, the root nodes are in the order they were declared in the file. Any added nodes are then appended to the list in the time order they were received at the browser. If the context was generated programmatically, the nodes are in the order they were received by the browser.

6.4.16 getRoutes

parameters:	SAIExecutionContext
returns:	SAIRoutes
errors:	SAI_INVALID_OPERATION_TIMING
	SAI_DISPOSED
events:	None
buffered:	No
external:	No

The getRoutes service gets the list of the current routes in the scene. The route listing returned will only be the top level routes. Routes contained within a PROTO definition or a prototype instance shall not be included in this list.

6.4.17 dynamicRouteHandling

parameters:	SAIExecutionContext, SAINode, SAIField, SAINode, SAIField, SAIAction
returns:	None
errors:	SAI_INVALID_OPERATION_TIMING
	SAI_INVALID_NODE

SAI_INVALID_FIELD
SAI_DISPOSED

events: None
buffered: Yes
external: No

The dynamicRouteHandling service is a request to process a route according to the action specified by the SAIAction value.

The parameter of type SAIAction specifies whether this should be an add or delete of this route. Other actions may be added, such as to query the existence of the nominated route. Actions defined by this part of ISO/IEC 19775 are described in Table 6.12. The SAINode/SAIField pair parameters are considered as defining the source field and destination fields for the route request.

Table 6.12 — dynamicRouteHandling SAIAction values

Service	Action Type
dynamicRouteHandling	AddRoute
	DeleteRoute

Route modification requests are to fit with the general event model scheme as defined in *4.4.8 Event model* in ISO/IEC 19775-1. The end of the event cascade is considered to be the cascade that is initiated by the application sending events into the X3D browser environment. Any new cascades generated as a result of the processing of the initial events shall not be considered for the determination of the event cascade.

If the action is to delete a route, and the route has previously been deleted, no error should be generated.

6.4.18 dispose

parameters: SAIExecutionContext
returns: None
errors: SAI_INVALID_OPERATION_TIMING
events: SAI_Browser_Shutdown
buffered: No
external: Yes

The dispose service specifies that the client has no further interest in the resource represented by this execution context. The browser may now take whatever action is necessary to reclaim any resources consumed by this execution context, now or at any time in the future. If this execution context has already been disposed, further requests have no effect.

6.5 Scene services

6.5.1 Introduction

A scene is an extension of the execution context services with additional services provided. The Scene services implementation shall include all of the services from 6.4 Execution context services, and include the following additional services.

6.5.2 getMetadata

TODO consider getMeta to avoid ambiguity with Metadata nodes

parameters:	SAIScene, SAIString
returns:	SAIString
errors:	SAI_INVALID_OPERATION_TIMING
events:	None
buffered:	No
external:	No

The `getMetadata` service returns an item of metadata from the scene that was specified using the META statement defined in 7.2.5.5 META statement in ISO/IEC 19775-1. Metadata specified in the META statement is represented as an SAIString key/value pair. Each key corresponds to exactly zero or one value.

Optionally, the browser may provide a subservice to discover the valid keys for this scene as part of this service.

Metadata defined by metadata nodes as defined in Part 1 of ISO/IEC 19775 are specifically different and can be manipulated using 6.6 Node services.

6.5.3 setMetadata

TODO consider SetMeta to avoid ambiguity with Metadata nodes

parameters:	SAIScene, SAIString, SAIString
returns:	None
errors:	SAI_INVALID_OPERATION_TIMING
events:	None
buffered:	Yes
external:	No

The `setMetadata` service inserts an item of metadata in the scene in the form of a META statement as defined in 7.2.5.6 META statement in ISO/IEC 19775-1. Metadata is represented as a SAIString key/value pair. Each key corresponds to exactly zero or one value. Setting an item with a key that already exists replaces the existing value. If the value is `NULL` for the given key, the META statement associated with that key is removed from the scene.

Metadata defined by metadata nodes as defined in ISO/IEC 19775-1 are specifically different and can be manipulated using 6.6 Node services.

6.5.4 namedNodeHandling

In addition to the capabilities described in 6.4.10 namedNodeHandling, the Scene services expand the capability to also work with exporting nodes. The definition is expanded to the following:

The `namedNodeHandling` service is a request to add, remove or change the node identified by the SAIString value where that name is considered to use the DEF, or IMPORT or EXPORT semantics. The add/remove/update shall be described by the first SAIAction parameter value. If the name already exists as a mapping, the

current mapping is replaced with the new mapping. When adding a new named node, it is not required to be a part of this scene.

The second SAIAction parameter value describes which of the DEF, IMPORT or EXPORT naming facilities shall be the target of this service request. This ensures that correct semantics are applied. If the action is to add and the name is already registered then SAI_NODE_IN_USE is generated. If the action is to replace or update, and the node is not already registered, the implementation may treat this as an add request.

The first SAIString value specifies a name with a node as it should be known in this scene. The name is not an intrinsic property of the node and this only serves as a mapping function.

The second argument provides an option depending on the action being undertaken. The SAINode value is a reference to the node that may be needed for verification of an EXPORT or DEF name addition. For adding IMPORTs, the second string shall be the exported node name in the DEF'd inline and the optional third string shall be the name to store it as in this scene (this corresponds with the Classic VRML syntax of *IMPORT inlined_def.export_name [AS import_name]*). For adding EXPORTs the second argument shall be the string, which is the optional name to export the node as. If the first SAIString does not describe a node marked with DEF, it shall generate a SAI_INVALID_NAME error.

6.5.5 rootNodeHandling

parameters:	SAIScene, SAINode, SAIAction
returns:	None
errors:	SAI_INVALID_OPERATION_TIMING SAI_INVALID_NODE SAI_DISPOSED SAI_IMPORTED_NODE SAI_NODE_IN_USE SAI_INSUFFICIENT_CAPABILITIES
events:	None
buffered:	Yes
external:	No

The `rootNodeHandling` service is a request to add and remove a root node of this scene.

The SAIAction parameter is used to indicate if the service request is an add or removal of the node. If the action is to remove and the node is not a known root node, this shall generate an error. If the action is to add the node, it shall be appended to the current list of root nodes. Table 6.13 defines the actions specified in this part of ISO/IEC 19775.

Table 6.13 — rootNodeHandling SAIAction values

Service	Action Type
rootNodeHandling	`AddRootNode`

	RemoveRootNode

Nodes are bound by the capabilities of the containing scene. No node shall be of greater capabilities than the scene's declared profile and additional components. SAI_INSUFFICIENT_CAPABILITIES shall be generated if the action is to add a node to the scene and that node requires greater capabilities than the scene permits.

If the action is to add a node and the node or any of its children is currently part of another scene, generate SAI_NODE_IN_USE.

If the action is to remove a node and the node is not a known value of this field, the request shall be silently ignored.

6.6 Node services

6.6.1 Introduction

The following services can be requested of an individual node. Each service requires an identifier for that node. After a request of an individual node to dispose of their resources, any further request made to a node service shall generate a disposed error.

Although not specified, all services are capable of throwing an SAI_CONNECTION_ERROR whenever a request is made if the session between the application and the browser has failed.

6.6.2 getTypeName

parameters:	SAINode
returns:	SAIString
errors:	SAI_DISPOSED
events:	None
buffered:	No
external:	No

The `getTypeName` service returns the name of the type of the referenced node. The type name is the name as specified in ISO/IEC 19775-1 where the node type is defined (see *Node index* in ISO/IEC 19775-1 for easy access to a node definition). If the node represents a PROTO node instance, the type name returned is the name of the PROTO declaration.

6.6.3 getType

parameters:	SAINode
returns:	SAINodeType
errors:	SAI_DISPOSED
events:	None
buffered:	No
external:	No

The getType service returns the type indicator for the referenced node. The type indicator is either the type defined for the basic node types in the X3D specification, or the PROTO type name if it is a prototyped node. This service is not required to be supported for a conforming implementation.

6.6.4 getField

parameters:	SAINode, SAIFieldName
returns:	SAIField
errors:	SAI_INVALID_OPERATION_TIMING
	SAI_INVALID_NAME
	SAI_DISPOSED
events:	None
buffered:	No
external:	No

The getField service returns a field identifier so that operations can be performed on the node properties. If the field requested is an inputOutput field, either the field name or the set_ and _changed modifiers may be used to access the appropriate form of the node as required. Access to fields is implementation dependent.

6.6.5 getFieldDefinitions

parameters:	SAINodeType
returns:	SAIFields
errors:	SAI_INVALID_OPERATION_TIMING
	SAI_INVALID_NAME
	SAI_DISPOSED
events:	None
buffered:	No
external:	No

The getFieldDefinitions service returns a list of all the field definitions of the referenced node. The definitions provide a limited form of the SAIField that has all the same services except the ability to read or write the value of the field for a specific node instance. This request returns the SAIField values as generic responses for every instance of this node rather than for a specific instance.

6.6.6 dispose

parameters:	SAINode
returns:	None
errors:	SAI_INVALID_OPERATION_TIMING
events:	None
buffered:	Yes
external:	No

The dispose node service indicates that the client has no further interest in the resource represented by this node. The browser may take whatever action is necessary to reclaim any resources consumed by this node, now or at any time in the future. If this node has already been disposed, further requests have no effect.

Disposing of a node does not remove the node from the scene graph (if it was inserted in the first place) but rather removes any local information per client to it. The underlying X3D node representation is only disposed if no other applications or scene graph structures contain references to this node. The responsibility and timing for this action is browser-implementation specific.

6.7 Field services

6.7.1 Introduction

The following are services that can be requested of individual fields of a node. If the node from which a field was retrieved has been disposed, field services are still permitted to operate providing that the field reference has been obtained before disposing of the node. If a call is made to a field service after requesting disposal of the field, a disposed error shall be generated.

Although not specified, all services are capable of throwing an SAI_CONNECTION_ERROR whenever a request is made if the session between the application and the browser has failed.

6.7.2 getAccessType

parameters:	SAINode, SAIField
returns:	SAIFieldAccess
errors:	SAI_INVALID_OPERATION_TIMING SAI_DISPOSED
events:	None
buffered:	No
external:	No

The `getAccessType` service returns the access type for the specified field of the referenced node.

6.7.3 getType

parameters:	SAINode, SAIField
returns:	SAIFieldType
errors:	SAI_DISPOSED
events:	None
buffered:	No
external:	No

The `getType` field service returns the type for the specified field of the referenced node.

6.7.4 getName

parameters:	SAINode, SAIField
returns:	SAIFieldName
errors:	SAI_DISPOSED
events:	None
buffered:	No

external:	No

If supported by the implementation, the `getName` field service returns the name of the field as it was requested from the node. If the service requested the *set_children* field of a grouping node, this shall return "set_children", but if a different request was for *children* on the same node, "children" shall be returned.

6.7.5 getValue

parameters:	SAINode, SAIField
returns:	SAIFieldValue
errors:	SAI_INVALID_OPERATION_TIMING
	SAI_INVALID_ACCESS_TYPE
	SAI_DISPOSED
events:	None
buffered:	No
external:	No

The `getValue` field service returns the value represented by the specified field as it exists in the world. This represents the current value of the field at the time of the request. If the request is made of a field that has a setValue request buffered through `BeginUpdate`, the value returned shall be the old value prior to the setValue request. The value of the field may be a node if the field represents an MFNode or SFNode.

All field types shall support the option to return a single value from multi-valued arrays.

6.7.6 setValue

parameters:	SAINode, SAIField, SAIFieldValue
returns:	None
errors:	SAI_INVALID_OPERATION_TIMING
	SAI_INVALID_ACCESS_TYPE
	SAI_IMPORTED_NODE
	SAI_DISPOSED
events:	None
buffered:	Yes
external:	No

The `setValue` field service sets the value of the specified field. Set requests shall obey the requirements as specified for buffered events services.

The value of the field may be an SAINode value if the field represents an MFNode or SFNode. It is permitted to send a `null` to a node or field in order to clear the value from that field. For example sending a `null` to the appearance inputOutput field of a Shape node as specified in *12 Shape component* in ISO/IEC 19775-1, shall result in the *appearance* field being cleared and set to the default value of `NULL`.

If the SAINode value is registered as an IMPORTed node in this file, it shall generate the SAI_IMPORTED_NODE error.

All field setting services implementations shall include the ability to set individual values. Fields that describe multi-value arrays shall also include the ability to append and remove items from the existing field.

6.7.7 registerFieldInterest

parameters:	SAINode, SAIField, SAIAction, SAIRequester
returns:	None
errors:	SAI_INVALID_OPERATION_TIMING SAI_INVALID_ACCESS_TYPE SAI_INSUFFICIENT_CAPABILITIES SAI_NODE_IN_USE SAI_DISPOSED
events:	SAIFieldEvent
buffered:	No
external:	No

The `registerFieldInterest` service nominates the requester as the receiver of all SAIFieldEvents. The act of making this service request itself does not imply any events shall be generated. Table 6.14 defines the actions specified in this part of ISO/IEC 19775.

Table 6.14 — registerFieldInterest SAIAction values

Service	Action Type
registerFieldInterest	`AddInterest`
	`RemoveInterest`

The parameter of type SAIRequester can be inferred from the source of the input and may not need to be part of the parameters.

The parameter of type SAIAction specifies whether this is a request to add interest in events or to remove interest in the events.

Which capabilities are permitted to be listened to are implementation dependent. For example, some implementations may permit listening to inputOnly values and outputOnly values while others will only permit listening to outputOnly values.

6.7.8 dispose

parameters:	SAIField
returns:	None
errors:	SAI_INVALID_OPERATION_TIMING
events:	None
buffered:	Yes
external:	No

The `dispose` field service indicates that the client has no further interest in the resource represented by this field. The browser may take whatever action is necessary to reclaim any resources consumed by this field, now or at any time in the future. If this field has already been disposed, further requests have no effect.

6.8 Route services

6.8.1. Introduction. Route services can be applied for handling ROUTE references wherever node services are allowed.

6.8.1 getSourceNode

parameters:	SAIRoute
returns:	SAINode
errors:	SAI_INVALID_OPERATION_TIMING
events:	None
buffered:	No
external:	No

The `getSourceNode` service returns the source node of the specified route.

6.8.2 getSourceField

parameters:	SAIRoute
returns:	SAIString
errors:	SAI_INVALID_OPERATION_TIMING
events:	None
buffered:	No
external:	No

The `getSourceField` service returns the name of the source field of the specified route.

6.8.3 getDestinationNode

parameters:	SAIRoute
returns:	SAINode
errors:	SAI_INVALID_OPERATION_TIMING
events:	None
buffered:	No
external:	No

The `getDestinationNode` service returns the destination node of the specified route.

6.8.4 getDestinationField

parameters:	SAIRoute
returns:	SAIString
errors:	SAI_INVALID_OPERATION_TIMING
events:	None
buffered:	No
external:	No

The `getDestinationField` service returns the name of the destination field of the specified route.

6.8.5 dispose

parameters:	SAIRoute
returns:	None
errors:	SAI_INVALID_OPERATION_TIMING
events:	None
buffered:	Yes
external:	No

The `dispose` route service indicates that the client has no further interest in the resource represented by this route. The browser may take whatever action is necessary to reclaim any resources consumed by this route, now or at any time in the future. If this route has already been disposed, further requests have no effect.

Disposing of a route does not remove the route from the scene graph (if it was inserted in the first place) but rather removes any local information per client to it. The underlying X3D node representation is only disposed of if no other applications or scene graph structures contain references to this route and the responsibility and timing for this action is browser implementation specific.

6.9 Prototype services

6.91. Prototype services can be applied for handling prototype references wherever Node services are allowed.

6.9.1 isExternproto

parameters:	SAIProtoDeclaration
returns:	SAIBoolean
errors:	SAI_DISPOSED
events:	None
buffered:	No
external:	No

The `isExternproto` service checks to see if the prototype declaration represents a PROTO or EXTERNPROTO. If it is an EXTERNPROTO, a `TRUE` value shall be returned, and PROTO declarations shall return `FALSE`.

6.9.2 createInstance

parameters:	SAIProtoDeclaration
returns:	SAINode
errors:	SAI_INVALID_OPERATION_TIMING SAI_INVALID_NODE SAI_DISPOSED
events:	None
buffered:	No
external:	No

The `createInstance` service creates a new instance from the declaration of either a PROTO or EXTERNPROTO. An EXTERNPROTO instance may fail with SAI_INVALID_NODE if the definition has not yet been loaded.

6.9.3 getFieldDefinitions

parameters:	SAIProtoDeclaration
returns:	

	SAIField(s)
errors:	SAI_INVALID_OPERATION_TIMING
	SAI_DISPOSED
events:	None
buffered:	No
external:	No

The `getFieldDefinitions` service returns a list of all the field definitions of the PROTO or EXTERNPROTO declaration. The definitions provide a limited form of the SAIField value that has all the same services except the ability to read or write the value of the field for a specific node instance.

6.9.4 checkLoadState

parameters:	SAIProtoDeclaration
returns:	SAILoadState
errors:	SAI_INVALID_NODE
	SAI_INVALID_URL
	SAI_DISPOSED
events:	None
buffered:	No
external:	No

The `checkLoadState` service checks on the current load state of the EXTERNPROTO definition. The state shall be one of NOT_STARTED, IN_PROGRESS, COMPLETE or FAILED. If this is called on a PROTO, a SAI_INVALID_NODE error shall be generated.

6.9.5 requestImmediateLoad

TODO consider getNodeType which returns type for first node in a prototype body declaration, if any is defined.

parameters:	SAIProtoDeclaration
returns:	None
errors:	SAI_INVALID_OPERATION_TIMING
	SAI_DISPOSED
events:	None
buffered:	No
external:	No

If the SAIProtoDeclaration value represents an EXTERNPROTO, the `requestImmediateLoad` service requests that the browser start immediate loading of that definition. If the definition is already loaded or the load is already in progress, this request shall be silently ignored.

6.10 Configuration services

6.10.1 Introduction

The services specified here allow an application to identify the configuration of the current world.

An SAIComponentDeclaration value specifies a component declaration containing the information specified by the COMPONENT statement as defined by 7.2.5.4

COMPONENT statement in ISO/IEC 19775-1. The services defined are the minimum required. An implementation may provide additional informational-only services.

An SAIProfileDeclaration value specifies a profile declaration containing the information specified by the PROFILE statement as defined by 7.2.5.3 PROFILE statement in ISO/IEC 19775-1 and by the profile definition contained in the annexes of ISO/IEC 19775-1. The services defined are the minimum required. An implementation may provide additional informational-only services.

6.10.2 getComponentName

parameters:	SAIComponentDeclaration
returns:	SAIString
errors:	None
events:	None
buffered:	No
external:	No

The `getComponentName` service returns the formal name of the specified component.

6.10.3 getComponentLevel

parameters:	SAIComponentDeclaration
returns:	SAIString
errors:	None
events:	None
buffered:	No
external:	No

The `getComponentLevel` service returns the support level specified for the component. When the SAIComponentDeclaration comes from the browser services, it shall represent the maximum level supported by the browser. When it comes from the scene services, the level represents the requested support level for that scene.

6.10.4 getProfileName

parameters:	SAIProfileDeclaration
returns:	SAIString
errors:	None
events:	None
buffered:	No
external:	No

The `getProfileName` service returns the formal name of the specified profile.

6.10.5 getProfileComponents

parameters:	SAIProfileDeclaration
returns:	SAIComponentDeclaration(s)
errors:	None
events:	None
buffered:	No

external:	No

The getProfileComponents service returns a list of SAIComponentDeclaration instances specifying the allowed support for each component of which the profile is comprised.

6.10.6 getProviderName

parameters:	SAIProfileDeclaration
returns:	SAIString
errors:	None
events:	None
buffered:	No
external:	No

The getProviderName service is an information-only service that returns an SAIString value containing the name of the person or company that implemented this profile.

6.10.7 getUnitCategory

parameters:	SAIUnitDeclaration
returns:	SAIString
errors:	SAI_INVALID_OPERATION_TIMING SAI_DISPOSED
events:	None
buffered:	No
external:	No

The getUnitCategory service returns the formal category of the specified unit declaration.

6.10.8 getUnitConversion

parameters:	SAIUnitDeclaration
returns:	SAIString
errors:	SAI_INVALID_OPERATION_TIMING SAI_DISPOSED
events:	None
buffered:	No
external:	No

The getUnitConversion service returns the conversion factor of the specified unit declaration.

6.10.9 getUnitName

parameters:	SAIUnitDeclaration
returns:	SAIString
errors:	SAI_INVALID_OPERATION_TIMING SAI_DISPOSED
events:	None
buffered:	No
external:	No

The `getUnitName` service returns the user-provided name of the specified unit declaration.

6.11 Services provided by script content

6.11.1 Introduction

When an author provides the executable content of a script, certain conventions shall be satisfied. This allows the browser to communicate status information unambiguously regardless of the type of content language. This clause defines services that are required to be defined by the individual language bindings in a manner such that script content may be informed in a consistent, unambiguous manner. Script content shall be required to run identically regardless of language used to author the content. In contrast to the other services specifications, the browser shall make these service requests of the user's code, and therefore the user code shall provide implementations of these, where necessary. All services are defined at the user's discretion and if the user does not define the service implementation, the browser shall silently continue.

6.11.2 Creation phase

During the creation phase, the script content is downloaded and an instance of the content created in the appropriate execution engine. Some content may require separate interpreters, while others may be created in the same address and execution space as the browser code (e.g., scripts created in the same language in which the browser itself was written). Apart from the instantiation process, which is language dependent, the browser shall not require any services.

6.11.3 Setup phase

During the setup phase, the browser provides the script with all of the run-time information that it will be able to use in the system.

6.11.3.1 setBrowser

parameters:	SAIBrowserRef
returns:	None
errors:	None
events:	None
buffered:	No
external:	No

The `setBrowser` service passes to the script implementation code the the SAIBrowserRef value to be used. There is no other way of acquiring the SAIBrowserRef during the lifetime of the script, so if the user code needs to know about it, it should store it now. This service shall be performed before any other service requests are made. The browser may call this service at any time between the creation phase and before the Initialize service request is made. The browser is not required to request it during the initialization process as defined *4.4.8*

Event model in ISO/IEC 19775-1 although that is time when the Browser implementers are encouraged to request this service.

6.11.3.2 setFields

parameters:	SAINode, SAIField(s)
returns:	None
errors:	None
events:	None
buffered:	No
external:	No

The `setFields` script service passes in the list of fields declared in this script node instance. It also passes in the external view of the containing script node so that the user may add and remove routes to the script directly. The SAIField instances represent all of the fields of a script, including the pre-defined fields. (See 4.8.3.4 Updating the scene graph for details on the field access restrictions). This request shall be performed between the setBrowser and initialize service requests.

6.11.3.3 initialize ? (formerly initialize)

parameters:	None
returns:	None
errors:	None
events:	None
buffered:	No
external:	No

The `initialize` script service provides notification to the script that all basic initialization has been completed and that the user code is active in the scene graph. At this point, the user code may access script field values and change the state of the script. The original name for this service is initialize, which is functionally equivalent and shall be supported.

6.11.4 Realized phase

6.11.4.1 prepareEvents

parameters:	None
returns:	None
errors:	None
events:	None
buffered:	No
external:	Yes

The `prepareEvents` service provides notification that the browser is about to start the event cascade processing in accordance with step 4 of the event execution model as described in *4.4.8.3 Execution model* in ISO/IEC 19775-1. All values changed during this call shall have the current time stamp, but events shall not be immediately propagated upon return from the user code. This service request shall be called every frame regardless of whether the containing node received any events. If the containing node provides directOutputs, these shall be passed immediately to the underlying nodes.

6.11.4.2 eventsProcessed

parameters:	SAIBrowserRef
returns:	None
errors:	None
events:	SAI_Browser_Shutdown
buffered:	No
external:	Yes

The `eventsProcessed` service provides notification that the current event cascade processing has finished and that the containing node is now allowed to make updates to the scene graph. This is useful for user code that wishes to be more efficient and only generate new events after a collection of field changes are received. Within a given frame, user code may have this service called more than once. User code cannot guarantee that all changes to the containing node will be received by this time and should take appropriate precautions. This service request shall only be called after the containing node has received one or more events in this timestamp. If the containing node has received no events in the current timestamp it shall be an error for the browser to request this service.

6.11.5 Disposed phase

6.11.5.1 shutdown

parameters:	None
returns:	None
errors:	None
events:	SAI_Browser_Shutdown
buffered:	No
external:	Yes

The `shutdown` service provides notification that the user code has been disposed of by the containing node. This may be due to the complete shutdown of the browser, the loaded world changing or the containing node changing the user code to another implementation. After this service request has been completed, user code will no longer be functional or executed.

6.12 Matrix services

6.12.1 Introduction

Matrix objects represent the standard mathematic matrix capabilities using double precision numbers and column-major order. All services here shall be interpreted using standard mathematical definitions of matrices.

Implementations shall provide matrices that are 3x3 and 4x4. They may define other orders of matrices. Implementations may also define additional convenience services in addition to this minimum subset; for example, the ability to individually access matrix elements. Implementations may allow direct access to the individual row and column values of the matrix.

In the following service definitions, the parameters describe single precision inputs. An implementation shall also provided overloaded definitions that include double precision input.

6.12.2 set

parameters:	SAIMatrix, SFVec3f, SFRotation, SFVec3f, SFRotation, SFVec3f
returns:	None
errors:	None
events:	None
buffered:	No
external:	No

The `set` matrix service sets the matrix to the new value calculated from the parameters. The parameters are defined to represent, in order: translation, rotation, scale, scaleOrientation, and center. If a value for a parameter is not specified, the default value for that parameter shall be the default value for the equivalent field of the Transform node defined in *10.4.4 Transform* in ISO/IEC 19775-1.

6.12.3 get

parameters:	SAIMatrix, SFVec3f, SFRotation, SFVec3f, SFRotation, SFVec3f
returns:	None
errors:	None
events:	None
buffered:	No
external:	No

The `get` service computes and returns the transformation values from the matrix. The parameters are defined to represent, in order: translation, rotation, scale, scaleOrientation and center.

6.12.4 inverse

parameters:	SAIMatrix
returns:	None
errors:	None
events:	None
buffered:	No
external:	No

The `inverse` service calculates the inverse of this matrix in place.

6.12.5 transpose

parameters:	SAIMatrix
returns:	None
errors:	None
events:	None
buffered:	No
external:	No

The `transpose` service transposes this matrix in place.

6.12.6 multiply

`parameters:`	SAIMatrix, SAIMatrix
`returns:`	None
`errors:`	None
`events:`	None
`buffered:`	No
`external:`	No

The `multiply` service multiplies the first matrix by the second matrix instance placing the result in the first matrix. Implementations shall define multiplication operations in both directions: left and right.

6.12.7 multiplyWithVector

`parameters:`	SAIMatrix, SFVec3f
`returns:`	SFVec3f
`errors:`	None
`events:`	None
`buffered:`	No
`external:`	No

The `multiplyWithVector` service multiplies this matrix and a vector together. Implementations shall define multiplication operations in both directions: left and right.

Extensible 3D (X3D)
Part 2: Scene access interface (SAI)

7 Conformance and minimum support requirements

7.1 Introduction and topics

7.1.1 Introduction

This clause defines the minimum required support for language bindings conforming to this part of ISO/IEC 19775.

7.1.2 Topics

Table 7.1 lists the topics for this clause.

Table 7.1 — Topics

- Table 7.5 — Specifications for SAI script content interaction
- Table 7.6 — Specifications for SAI utility services

7.1.3 Objectives

This clause addresses conformance of X3D browsers that provide a scene authoring interface (SAI).

Add authoring

The primary objectives of the specifications in this clause are:

a. to promote interoperability by eliminating arbitrary subsets of ISO/IEC 19775;
b. to promote extensibility within a well-defined environment;
c. to promote uniformity in the development of conformance tests;
d. to promote consistent results across X3D browsers;
e. to facilitate automated test generation.

7.1.4 Scope

Conformance is defined for language bindings to this specification and therefore X3D Browsers and applications that use the facilities provided by the Scene Authoring Interface language-specific bindings as specified in ISO/IEC 19777.

Due to the abstract nature of this specification, it is not possible to specify the conformance tests of individual language bindings of the specification. Separate Conformance sections shall be provided within each language binding that provide the necessary information to implement language-specific binding conformance tests.

A concept of base profile conformance is defined to ensure interoperability of X3D applications and X3D browsers. Base profile conformance is based on a set of limits and minimal requirements. Base profile conformance is intended to provide a functional level of reasonable utility for X3D language bindings while limiting the complexity and resource requirements of X3D browsers. Base profile conformance may not be adequate for all uses of the SAI.

7.2 Conformance

7.2.1 Conformance to this part of ISO/IEC 19775

A X3D browser is only conformant to this part of ISO/IEC 19775 if it also conformant to the current profile as specified in ISO/IEC 19775-1. In addition, the following conditions shall be met:

a. Requests of the Browser services shall conform exactly to the behaviour as specified in this part of ISO/IEC 19775.
b. Where a browser is required to read and parse X3D content, it shall be able to handle any X3D file that conforms to the profile being supported as

defined in ISO/IEC 19775-1 or separate specification for such profiles.

c. Only nodes in the top level world (the file defined by the `getWorldURL` service) that are named with the DEF construct shall be visible to the `getNode` browser service request.

d. There shall be no difference in treatment of events that result from an external service request compared to events generated within the X3D browser environment. That is, externally generated event cascades shall have no favoured treatment.

7.2.2 Language bindings

A language binding to this part of ISO/IEC 19775 is conforming if:

a. It implements the services defined in this specification including return values, error conditions and asynchronous events.

b. The services implemented conform to the required functionality for the subject profile.

c. It provides sections outlining conformance and minimum requirements of implementations of that binding.

7.3 Minimum Requirements

7.3.1 Requirements for implementations

All profiles supporting the authoring component shall support the Level 1 functionality as defined in this section. There are two levels of conformance, that of the language binding and the browser implementation.

~More-thorough requirements for

A language binding shall at a minimum implement the services required for the profile to be supported. It may also define its own optional set of minimum requirements that are no less than the requirements of this section. A browser implementation shall, in addition to the base support for the subject profile, also support the minimum capabilities defined for the subject profile. In general, the browser requirements are specified as general requirements that deal with specific language issues. This level of conformance is sufficient for author-defined Script nodes and also programmatic construction of X3D Where a browser implements two or more language bindings with different levels of minimum requirements, the browser shall support the minimum requirements Models for each language separately. Therefore, if one language has higher requirements, the lower requirements of the other shall not be used.

7.3.2 Supporting internal and external interactions

Although external interactions use a superset of the services defined for internal interactions, language bindings to the abstract specification are not required to implement both. They may choose one or the other or both. (It is strongly recommended that a binding support both.)

A conformant browser implementation is not required to provide both internal and external implementations of a given language binding. It shall be possible for a

browser to support internal interactions only with language A and external interactions only with language B. In addition, the browser may choose to support only internal interactions or only external interactions. There is no requirement to support both internal and external interactions.

7.3.3 Level 1

Table 7.2 through Table 7.6 define the minimum requirements for a base profile. The first column specifies the item for which conformance is being defined. These refer to the services as defined by this specification. The second column specifies the requirements for a language binding specification of that item. The third column specifies the requirements of a browser implementation of that service. SAI implementations shall throw an SAIError (such as SAI_NOT_SUPPORTED) if a capability is not supported.

For all these requirements, it shall be assumed that the language binding shall provide complete implementation of all the parameters required for the individual services. Language bindings of data types may be implemented as primitive types in the target language rather than as separate data types. Table 7.2 through Table 7.6 indicate where this is permitted.

A browser conformant to this level shall support bindings to the X3DNode and X3DMetadataObject abstract representations along with all field types.

Table 7.2 — Specifications for SAI data type implementations

Item	Binding support	Minimum browser support
SAIAction	Full support as required by the individual service request	As defined by the language
SAIBoolean	Full Support	Full Support
SAIBrowserApp	Full Support if createBrowser supported. Not required if only getBrowser supported.	Optional (based on getBrowser/createBrowser requirements)
SAIBrowserName	Primitive Type	Full Support
SAIBrowserRef	Full Support	Full Support
SAIBrowserVersion	Primitive type	Full Support
SAIComponentDeclaration	Primitive type describing at least name and level	Full Support
SAIComponent	Primitive type	Full Support
SAIEncoding	Specifies the type of	Full Support

	encoding.	
SAIExecutionContext	Provides access to a subscene.	Full Support
SAIFieldAccess	Separate data types for the four types defined by ISO/IEC 19775-1.	Full Support
SAIFieldDeclaration	Provide information on access type, data type and name	Full Support
SAIField	Full Support	Full Support
SAIFieldName	Primitive type	Full Support
SAIFieldType	Separate data types for all types defined in 5 Field type reference of ISO/IEC 19775-1.	Full Support
SAIFieldValue	Primitive type as appropriate to the given field. Where field is an SF/MFNode shall be SAINodeID	Number of values for setting and getting as defined in the applicable profile as defined in ISO/IEC 19775-1.
SAIFrameRate	Primitive Type	Full Support
SAILayerID	Primitive Type	Full Support if the Layering component is supported.
SAILoadState	Primitive Type	Full Support
SAIMatrix	Primitive Type	Full Support
SAINavSpeed	Primitive Type	Full Support
SAINode	Full Support	Full Support
SAINodeType	Primitive type	SAIString representation of the node name
SAIParameterList	As required by langauge and service definition.	Dependent on language and browser implementation
SAIProfileDeclaration	Description of at least name and components used in the profile	Full Support
SAIPropertyList	Primitive type	5 key-value pairs. Values dependent on language bindings

SAIProtoDeclaration	Full Support	n/a
SAIRequester	Full Support	Full Support
SAIRoute	Full Support	Full Support
SAIScene	Full Support	Full Support
SAIScript	Full Support	Full Support
SAIScriptImplementation	Full Support	Full Support
SAIStream	Full Support	Full Support
SAIString	Full Support	Full Support
SAIUnitDeclaration	Full Support	One entry for each type of unit category.
SAIURL	Both URL and URN support	As specified in the applicable profile defined in ISO/IEC 19775-1 for all *url* fields
SAIError	Separate types for each error condition that may occur as defined in 5.3 Error Types.	Generate error conditions as appropriate

Table 7.3 — Specifications for SAI browser implementations

Item	Binding support	Minimum internal support	Minimum external support
Establishing a connection			
getBrowser, createBrowser	At least one of getBrowser or createBrowser services shall be provided.	N/A	At least one method of connection with a browser shall be provided. Unsupported connection methods shall throw an error. Ignore *SAIParameterList*
Browser services			
getName	Shall provide	Return NULL if not supported	Return NULL if not supported
getVersion	Shall provide	Return NULL if not supported.	Return NULL if not supported
getCurrentSpeed	Shall provide	Return 0.0 if	Return 0.0 if not supported

		not supported	
getCurrentFrameRate	Shall provide	Return 0.0 if not supported	Return 0.0 if not supported
getSupportedProfiles	Shall provide	Full Support	Full Support
getProfile	Shall provide	Full Support	Full Support
getSupportedComponents	Shall provide	Full Support	Full Support
getExecutionContext	Shall provide	Full Support	Full Support
createScene	Shall provide	Support creating scenes for the same profiles and components as used by data encodings.	
replaceWorld	Full support	Full support	Full support
importDocument	Shall provide	Return NULL if not supported	Return NULL if not supported
loadURL	Shall provide	Full support. Ignore *SAIPropertyList* parameter values.	Full support. Ignore *SAIPropertyList* parameter values.
setDescription	Shall provide	No restriction	No restriction
createX3DFromString	Shall provide	Support File Limits as specified in the applicable profile defined in ISO/IEC 19775-1.	
createX3DFromStream	Provision dependent of language capabilities for creating raw I/O streams	Support File Limits as specified in the applicable profile as defined in ISO/IEC 19775-1.	
createX3DFromURL	Shall provide	Support File Limits as specified in the applicable profile as defined in ISO/IEC 19775-1.	
updateControl	SAIActions of start buffering and end buffering	N/A	Full support
registerBrowserInterest	SAIActions of add and remove interest.	Events for initialization, shutdown, invalid URLs and connection lost.	
getRenderingProperties	Shall provide	No restrictions	No restrictions

getBrowserProperties	Shall provide	No restrictions	No restrictions
setBrowserOptions	Shall provide	No restrictions	No restrictions
changeViewpoint	Shall provide	No restrictions	No restrictions
print/println	Shall provide	No restrictions	No restrictions
dispose	Shall provide	No restrictions	No restrictions

Table 7.4 — Specifications for SAI general services

Item	Binding Support	Minimum Browser Support
Execution context services		
getSpecificationVersion	Full Support	Full Support
getEncoding	Shall provide	Full Support
getProfile	SAIProfileDeclaration	Full support.
getComponents	SAIComponentDeclarations	Full Support
getUnits	SAIUnitDeclarations	Full Support
getWorldURL	Shall provide	Full Support
getNode	Full Support	Full Support
createNode	Full Support	Full Support
createProto	Full Support	Full Support
namedNodeHandling	SAIActions of add and delete nodes and imports	Full support.
getProtoDeclaration	Shall provide	Full Support
protoDeclarationHandling	SAIActions of add and delete PROTO	Full Support
getExternProtoDeclaration	Full Support	Full Support
externprotoDeclarationHandling	SAIActions of add and delete EXTERNPROTO	Full support.
getRootNodes	Shall provide	Full Support
getRoutes	Shall provide	Full Support
dynamicRouteHandling	SAIActions of add and	Full support.

	delete route	
dispose	Shall provide	No restrictions
Scene services		
getMetaData	Shall provide	Full Support
setMetaData	Shall provide	Full Support
namedNodeHandling	SAIActions of add and delete exports	Full support
rootNodeHandling	SAIActions of add and delete nodes	Full support
Node services		
getTypeName	Shall provide	Full support
getType	Shall provide	no restrictions
getField	Full Support	All fields shall be accesible dependent on access rules for internal and external interactions and node lifecycle..
getFieldDefinitions	Full Support	Full Support
dispose	Shall provide	No restrictions
Field services		
dispose	Full support	Full support
getAccessType	Shall provide	Full support
getType	see SAIFieldType	Full support
getName	Full Support	Field name without *set_* or *_changed* modifiers
getValue	get1Value not required	Full Support
setValue	set1Value not required	Full Support. Where fields are MF fields, minimum number of values to be supported as specified in the applicable profile defined in ISO/IEC 19775-1.
registerFieldInterest	SAIActions of add and remove interest	As per supported langauge binding(s).

	outputOnly and the output side of inputOutput fields shall be supported	
Route services		
dispose	Full support	Full support
getSourceNode	Full support	Full Support
getSourceField	Full support	Full Support
getDestinationNode	Full support	Full Support
getDestinationField	Full support	Full Support
Prototype services		
isExternProto	Full support	Full Support
createInstance	Full support	Full Support
getFieldDefintions	Full support	Full Support
checkLoadState	Full support	Full Support
requestImmediateLoad	Full support	Full Support
Configuration services		
getComponentName	Full support	Full Support
getComponentLevel	Full support	Full Support
getProfileName	Full support	Full Support
getProfileComponents	Full support	Full Support
getProviderName	Full support	Full Support
getUnitCategory	Full support	Full Support
getUnitConversion	Full support	Full Support
getUnitName	Full support	Full Support

Table 7.5 — Specifications for SAI script content interaction

Item	Binding support	Minimum browser support
setBrowser	Full support	Full Support

setFields	Full support	Full Support
initialize	Full support	Full Support
prepareEvents	Full support	Full Support
eventsProcessed	Full support	Full Support
shutdown	Full support	Full Support

Table 7.6 — Specifications for SAI utility services

Item	Binding support	Minimum browser support
Matrix	At least 3x3 and 4x4 sizes	Full Support

7.3.4 Level 2

A browser conformant to Level 2 shall support everything in Level 1 and shall support all abstract node types derived from X3DNode for the components implemented in the browser. The browser shall also support all additional objects that may be introduced by the implementation.

7.3.5 Level 3

A browser conformant to Level 3 shall support all of the requirements of Level 2 and also support binding specific interfaces for each concrete node representation.

TODO. Ensure that all added or revamped services in prior clauses of this specification are included in the tables for Clause 7, appropriately.

Extensible 3D (X3D)
Part 2: Scene access interface (SAI)

Annex A

(informative)

VRML scripting backwards compatibility

A.1 Introduction and topics

A.1.1 Introduction

This annex provides a detailed description of how an X3D compliant application may provide support for backwards compatibility for scripting code originally designed and implemented for *ISO/IEC 14772-1 Virtual Reality Modeling Language (VRML) Functional specification and UTF-8 encoding*. It is provided for backwards-compatibility purposes only and shall not be required for any browser to implement if they do not conform to part 1 of ISO/IEC 14772.

The VRML event and scripting model had a number of flaws and incompatibilities even between languages. It is strongly recommended that this annex not be implemented by browsers that do not claim to support ISO/IEC 14772-1 in addition to ISO/IEC 19775-1.

A.1.2 Topics

Table A.1 lists the topics for this annex.

Table A.1 — Topics

A.2 Concepts

A.2.1 Introduction

A.2.1.1 Requirements for backwards compatibility

This annex specifies the requirements for browsers that wish to conform to this specification and also support the ability to run scripts written for the programmatic interfaces and event model semantics defined in ISO/IEC 14772-1, within an X3D environment. That is, the URL in the Script node specified in *29.4.1 Script* in ISO/IEC 19775-1 defines content that uses one of the interfaces defined in this part of ISO/IEC 19775.

A browser shall not be conformant to this specification if it only supports scripting interface defined in ISO/IEC 14772-1. A conformant browser to this Annex shall also support the full requirements of this part of ISO/IEC 19775 including all required language bindings.

A.2.1.2 Differences between VRML and X3D

The main difference between ISO/IEC 14772-1 and this part of ISO/IEC 19775 is the definition of the event model. ISO/IEC 14772-1 leaves many decisions up to the browser implementer and therefore a lot of Script node content can be incompatible. The major issue is dealing with the way scene graph changes are propagated when the user code writes to the field. In the Java language annex values need to be delivered immediately, yet the ECMAScript language annex said they are to be deferred until the user code has exited execution. Such incompatibilities mean supporting direct backwards compatibility is an optional-only feature.

A.2.2 Behaviour for unsupported scripting

Determination of the supported script capabilities shall follow the rules specified in *9 Networking Component* in ISO/IEC 19775-1. The browser shall attempt to load the URIs in the required order, determining whether an individual script file is supported. Rejection of unsupported script types shall be based on the rules defined in the corresponding annexes of ISO/IEC 14772-2. A browser shall not preferentially support one specification over another beyond the preference order defined by the script node definition. For example, if the user content defines a ISO/IEC 14772-1 conformant script code in the *url* field before a ISO/IEC 19775-1 conformant script code, and the browser implementation supports backwards compatibility in a language as defined in this Annex, then the ISO/IEC 14772-1 script shall be executed.

When user content defines internal interaction code that includes ISO/IEC 14772-1-conformant scripts and the browser implementation does not support backwards compatibility, or does not support backwards compatibility in that language, the browser shall ignore that script and move to the next item in the *url* listings. This is the same behaviour as not being able to locate a file or the code is not in a language supported by the browser.

A.2.3 Behaviour on encountering exposedFields

ISO/IEC 14772-1 does not permit the use of read and write fields (i.e., exposedFields). The use of such fields is permitted in this part of ISO/IEC 19775. It is possible, though a highly discouraged practice, to define a scripting node with an exposedField and define the user code conformant to ISO/IEC 14772-1.

In this situation, the browser shall treat the exposedField as a separate set of eventIn, eventOut and field objects. When the exposedField is written from the event model, the user code shall be notified like a normal eventIn notification. After this point, the rules for reading and writing the value of the exposedField semantics defined in 4.8.3.8 inputOutput fields and the containing node shall be followed.

A.3 ECMAScript language binding

A.3.1 Requirements

If a browser intends to support ISO/IEC 14772 backwards compatibility for the ECMAScript language in the script node, it shall do so in conformance with *Annex C ECMAScript language binding* in ISO/IEC 14772-1.

A.3.2 Determination of required specification

A browser shall determine which version of the specification is supported through the use of the protocol definition or MIME type given with the external file. No other indicators shall be used.

A.3.3 Supported script URLs

A.3.3.1 Inline script definition

Browsers supporting the ECMAScript backwards compatibility shall support the use of inlined script nodes through the use of the customized protocol definition *javascript*. The support specified in *7 Conformance and minimum support requirements* in ISO/IEC 14772-1 is required in addition to the other required protocols for the ECMAScript specification as specified in *Annex C ECMAScript scripting reference* in ISO/IEC 14772-1. An example of the inlined script definition is:

```
Script {
    url "javascript: function foo() { ... }"
}
```

The url field may contain multiple URL's referencing either a remote file or in-line code as shown in the following example:

```
Script {
    url [
        "http://foo.com/myScript.js",
        "javascript: function foo( ) { ... }"
    ]
}
```

The use of the *javascript* protocol shall require the browser to support the objects and semantics defined in the Annex A specification. It shall be an error for a browser that conforms to this part of ISO/IEC 19775 to support the *javascript* protocol with script content that uses Objects defined in Annex A of this part of ISO/IEC 19775.

The use of the *ecmascript:* protocol shall indicate the script conforms to Annex A of this part of ISO/IEC 19775. A browser shall not provide ISO/IEC 14772-1-defined objects to a script that uses the *ecmascript:* protocol.

A.3.3.2 MIME types

The MIME type for ISO/IEC 14772-1 ECMAScript source code is defined as follows:

```
application/javascript
```

For backwards compatibility with old web servers, it is recommended browsers also support the following mime type:

```
application/x-javascript
```

The use of the `application/ecmascript` or `application/x-ecmascript` MIME types shall indicate the script conforms to Annex A of this specification. A browser shall not provide VRML-defined objects to a script that uses these MIME types.

A.4 Java language binding

A.4.1 Requirements

If a browser intends to support ISO/IEC 14772-1 backwards compatibility for the Java language in the script node, they shall do so in conformance with *Annex B Java platform scripting reference* in ISO/IEC 14772-1.

A.4.2 Determination of required specification

Since a mechanism is not defined formally!

As it is impossible to predetermine which set of classes and interfaces a Java class file implements through the use of the URL protocol or MIME type, a browser shall determine which version of the specification is supported by examining the base class or interfaces implemented by the user code. Determination of this may be made through the language introspection capabilities or using the `instanceof` operator.

A Java script that claims to support ISO/IEC 14772-1 shall extend the base class `vrml.node.Script`. If a script class file extends the ISO/IEC 14772-1 base class and implements the `org.web3d.x3d.sai.X3DScriptImplementation` interface then the browser shall use the X3D SAI semantics and execution and is not required to support ISO/IEC 14772-1 or objects that conform to ISO/IEC 14772-1. It is recommended that a browser issue a warning when it detects such a situation.

Explicitly describing region/cap