EUROPEAN TELECOMMUNICATION STANDARD

ETS 300 714

May 1997

Source: ETSI TC-MTA

Reference: DE/MTA-001045

formerly DE/TE-01045

ICS: 33.020

Key words: API, MHEG, multimedia

Terminal Equipment (TE); Application Programming Interface (API) for the manipulation of Multimedia and Hypermedia information objects

ETSI

European Telecommunications Standards Institute

ETSI Secretariat

Postal address: F-06921 Sophia Antipolis CEDEX - FRANCE
Office address: 650 Route des Lucioles - Sophia Antipolis - Valbonne - FRANCE
X.400: c=fr, a=atlas, p=etsi, s=secretariat - **Internet:** secretariat@etsi.fr

Tel.: +33 4 92 94 42 00 - Fax: +33 4 93 65 47 16

Contents

Blank page

Foreword

This European Telecommunication Standard (ETS) has been produced by the Terminal Equipment (TE) Technical Committee of the European Telecommunications Standards Institute (ETSI).

Introduction

This ETS specifies the abstract Application Programming Interface (API) for the manipulation of multimedia and hypermedia information objects, i.e. the API that shall be provided by Multimedia and Hypermedia Experts Group (MHEG) engines for their control by MHEG applications.

This ETS is part of a broader standardisation framework that specifies the usage of MHEG so that interoperable equipment can be effectively developed to support multimedia information services and applications. This implies:

- specifying additional constraints on the use of MHEG objects within distributed systems and applications using telecommunication networks;

- defining APIs that building blocks of architectures using MHEG should provide;

- defining MHEG profiles complementing the MHEG-1 standard by specifying restrictions on the coded representation and specifying the complete required behaviour of an MHEG engine that should be supported for a given category of applications and/or terminal equipment;

- defining an MHEG script interchange representation;

- defining end-to-end protocols for multimedia/hypermedia information services using MHEG;

- specifying conformance testing procedures for these standards.

Functional and technical requirements of this ETS have been described in ETR 225 [4].

Transposition dates	
Date of adoption:	18 April 1997
Date of latest announcement of this ETS (doa):	31 August 1997
Date of latest publication of new National Standard or endorsement of this ETS (dop/e):	28 February 1998
Date of withdrawal of any conflicting National Standard (dow):	28 February 1998

Blank page

1 Scope

Multimedia and Hypermedia Experts Group (MHEG) Part 1 (ISO/IEC DIS 13522-1 [1]) is a generic standard, which specifies the coded representation of interchanged multimedia/hypermedia information objects (MHEG objects). These so-called MHEG objects are handled, interpreted and presented by MHEG engines.

This European Telecommunication Standard (ETS) specifies the abstract Application Programming Interface (API) for the manipulation of multimedia and hypermedia information objects, i.e. the API that shall be provided by MHEG engines for their control by MHEG applications.

This API meets the following requirements:

- it is independent of the programming language used for the MHEG application;
- it is independent of the underlying operating system;
- it is independent of the mechanism used for interchanging information between the API user (i.e. MHEG application) and the API provider (i.e. MHEG engine, the messages that are exchanged as the result of triggering API primitives);
- it is independent of the actual encoding of these messages;
- it is generic and meant to cover all application requirements;
- it is conformance testable;
- it aims to be as easy as possible to implement.

2 Normative references

This ETS incorporates by dated and undated reference, provisions from other publications. These normative references are cited at the appropriate places in the text and the publications are listed hereafter. For dated references, subsequent amendments to or revisions of any of these publications apply to this ETS only when incorporated in it by amendment or revision. For undated references the latest edition of the publication referred to applies.

[1] ISO/IEC DIS 13522-1 (1994): "Information technology - Coding of Multimedia and Hypermedia Information - Part 1: MHEG object representation - Base Notation".

[2] ISO/IEC 9646 Parts 1 to 5 (1991): "Information Technology - Open Systems Interconnection - Conformance testing methodology and framework".

[3] ETR 173 (1995): "Terminal Equipment (TE); Functional Model for Multimedia Applications".

[4] ETR 225 (1995): "Terminal Equipment (TE); Application Programmable Interface (API) and script representation for MHEG; Requirements and framework".

[5] (Reserved).

[6] (Reserved).

[7] ITU-T Recommendation I.113 (1994): "Vocabulary of terms for broadband aspects of ISDN".

[8] ITU-T Recommendation I.112 (1993): "Vocabulary of terms for ISDNs".

[9] CCITT Recommendation Q.9 (1990): "Vocabulary of switching and signalling terms".

[10] ISO/IEC 14750-1 Working Draft: "CORBA IDL as an Interface Definition Language for ODP Systems".

3 Definitions and abbreviations

3.1 Definitions

For the purposes of this ETS, the following definitions apply:

NOTE: Due to the particular nature of this ETS, some of the words and expressions used in this ETS come from the "telecommunication services" standards glossary, while others come from the "software technology" standards glossary. This leads to words whose meaning vary according to the context, i.e. the expression within which they are used. For this reason, many of these expressions are defined in this subclause.

Should any ambiguity occur, definitions of the following standards would apply, in decreasing order:

- ISO/IEC DIS 13522-1 [1];
- any other standard part of ISO/IEC 13522 [1];
- ITU-T Recommendation I.113 [7];
- ITU-T Recommendation I.112 [8];
- CCITT Recommendation Q.9 [9].

Application Programming Interface (API): A boundary across which a software application uses facilities of programming languages to invoke software services. These facilities may include procedures or operations, shared data objects and resolution of identifiers.

function family: A cluster of functional MHEG API requirements consisting of functions with related semantics and applying to the same type of target.

hypermedia: The ability to access monomedia and multimedia information by interaction with explicit links.

interactive service: A service which provides the means for bidirectional exchange of information between users or between users and hosts. Interactive services are subdivided into three classes of services: conversational services, messaging services and retrieval services (ITU-T Recommendation I.113 [7]).

local application: A piece of software which is part of the (telecommunication) application and is running on the considered equipment.

MHEG API: The API provided by an MHEG engine to MHEG applications for the manipulation of MHEG objects, as defined in this ETS.

MHEG application: A piece of software which uses the MHEG API. An MHEG application is therefore a client of an MHEG engine.

MHEG engine: A process or a set of processes that interpret MHEG objects encoded according to the encoding specifications of ISO/IEC DIS 13522-1 [1]: Abstract Syntax Notation One (ASN.1) for Part 1, Standard Generalized Markup Language (SGML) for Part 2.

MHEG using application: An application which involves the interchange of MHEG objects within itself or with another application.

multimedia and hypermedia application: An application which involves the presentation of multimedia information to the user and the interactive navigation across this information by the user.

Multimedia And Hypermedia Information Retrieval Services (M&HIRS): A generic set of services which provide users with the capability to access and interchange multimedia and hypermedia information.

multimedia application: An application which involves the presentation of multimedia information to the user.

multimedia: The property of handling several types of representation media.

primitive: One of the basic entry points provided by a provider module to any user module to enable the user module to access the software service(s) supplied by the provider module.

software application: A piece of software answering a set of user's requirements and for use by a computer user.

software service: A set of functions provided by a (server) software or system to a client software or system, usually accessible through an application programming interface.

telecommunication application: A set of a user's requirements (CCITT Recommendation Q.9 [9]).

telecommunication service: That which is offered by an administration to its customers in order to satisfy a specific telecommunication requirement (ITU-T Recommendation I.112 [8]).

terminal application: A piece of software running on the terminal and performing that part of the processing required to make the terminal appropriate for user access to the application. The terminal application is usually the "master" module in the terminal.

user: A person or machine delegated by a customer to use the services and/or facilities of a telecommunication network (ITU-T Recommendation I.112 [8]).

action object: An object that provides operation on objects: e.g. to change their attributes or states.

channel: A logical space in which rt-components are positioned for final presentation. Channels are mapped by the MHEG engine to physical devices such as screen windows or loudspeaker for making the rt-objects within them perceivable by the user.

component object: An abstraction which represents objects of Content or Composite type.

composite object: A list of Composition Elements grouped for presentation. The presentation of a Composite object consists of the presentation of its Composition Elements.

container object: A means to group objects without specifying specific relationships.

content object: Encoded generic value, media or non-media data.

descriptor object: A structure for the interchange of resource information about a single or a set of other interchanged objects.

IDL attribute: An identifiable association between an object and a value. An attribute A is made visible to clients as a pair of operations: get_A and set_A. Read only attributes only generate a get operation.

IDL class: An implementation that can be instantiated to create multiple objects with the same behaviour. An object is an instance of a class. Types classify objects according to a common implementation.

IDL data type: A categorisation of values operation arguments, typically covering both behaviour and representation (i.e., the traditional non-Object Oriented (OO) programming language notion of type).

IDL instance: A object is an instance of an interface if it provides the operations, signatures and semantics specified by that interface. An object is an instance of an implementation if its behaviour is provided by that implementation.

IDL object: A combination of state and a set of methods that explicitly embodies an abstraction characterised by the behaviour of the relevant requests. An object is an instance of an implementation and an interface. An object models a real-world-entity, and it is implemented as a computational entity that encapsulates state and operations (internally implemented as data and methods) and responds to requester services.

IDL operation: A service that can be requested. An operation has an associated signature, which may restrict which actual parameters are valid.

Interface Definition Language (IDL): A language that specifies types and objects by specifying their interface. It provides a conceptual framework for describing the objects.

link object: An object which defines spatio-temporal relationships between other objects.

macro object: An object that provides the facility to replace the parameters in frequently used action objects.

mh-object: MhObjects (and their subtypes) match form b) objects as defined in ISO/IEC DIS 13522-1 [1], subclause 6.2.4, i.e. objects available to the MHEG engine.

MHEG elementary action: An attribute of the MHEG Action class which instructs an object to perform a certain operation: e.g. to change one of its attributes or states.

MHEG entity: Any MHEG object, rt-object, content data, script data, socket, channel or other construction identified or referred to in ISO/IEC DIS 13522-1 [1].

MHEG interpretation service: The service takes interchanged MHEG objects and messages issued by the presentation system as an input. It analyses this data in order to trigger the presentation of content data according to its semantics.

MHEG object handling service: This facility physically creates, handles and maintains intermediate data structures necessary to implement a client access to the MHEG objects and their contents.

MHEG object: A coded representation of an MHEG object class instance.

model object: Object instanciations from the MHEG classes.

notification: A primitive issued by the server on its own initiative to forward information to the client.

request: A primitive issued by the client on its own initiative to forward information to the server.

response: A primitive issued by the server as a reply to a request to forward information to the client.

rt-object: RtObjects (and their subtypes) match form c objects as defined in ISO/IEC DIS 13522-1 [1], subclause 6.2.4, i.e. instances of MhObjects available to the presentation process.

script object: Object that provides the structure to interchange script data in a specified encoded form.

socket: An element of an rt-composite. Rt-components are plugged into sockets. Different types of sockets are defined depending on the rt-component plugged into the socket:

- empty socket, i.e. a null rt-component is plugged;
- presentable socket, i.e. an rt-content or an rt-multiplexed content is plugged;
- structural socket, i.e. an rt-composite is plugged.

3.2 Abbreviations

For the purposes of this ETS, the following abbreviations apply:

AP	Attachment Point
API	Application Programming Interface
ASN.1	Abstract Syntax Notation One
C1	State period for a Channel: "non available"
C2	State period for a Channel: "processing"
C3	State period for a Channel: "available"
C4	State period for a Channel: "processing"
CGS	Channel Generic Space
CGSU	Channel Generic Space Unit
CORBA	Common Object Request Broker Architecture
EBNF	Extended Backus Naur Form
GSR	Generic Spatial Ratio
GTU	Generic Temporal Unit
IDL	Interface Definition Language (as defined in ISO/IEC 14750-1 [10])
IOGTR	Initial Original generic space Generic Temporal Ratio
IOV	Initial Original audible Volume
MCU	Multipoint Control Unit
mh	multimedia/hypermedia
MHEG	Multimedia and Hypermedia information coding Experts Group
MPEG	Moving Picture Experts Group
O1	State period for an mh-object: "not ready"
O2	State period for an mh-object: "processing"
O3	State period for an mh-object: "ready"
O4	State period for an mh-object: "processing"
OGS	Original Generic Space
OGSU	Original generic space Generic Spatial Unit
OGTR	Original generic space Generic Temporal Ratio
OGTU	Original generic space Generic Temporal Unit
OO	Object Oriented
OV	Original Volume
PRGS	Parent Relative Generic Space
PS	Perceptible Size
PSAP	Perceptible Size Attachment Point
R1	State period for an rt-object: "not available"
R2	State period for an rt-object: "processing"
R3	State period for an rt-object: "available"
R4	State period for an rt-object: "processing"
RGS	Relative Generic Space
RGSU	Relative generic space Generic Spatial Unit
RGTU	Relative generic space Generic Temporal Unit
rt	run-time
SGML	Standard Generalized Markup Language
SIR	Script Interchange Representation
SSU	Service Support Unit
VCR	Video Cassette Recorder
VD	Visible Duration
VS	Visible Size
VSEAP	Visible Size External Attachment Point
VSGS	Visible Size Generic Space
VSGSU	Visible Size Generic Spatial Unit
VSIAP	Visible Size Internal Attachment Point

4 Conformance

An implementation of this ETS is an MHEG engine implementation which provides client MHEG applications with one or several language bindings of the abstract API defined in this ETS.

An application of this ETS is an MHEG application which uses a language binding of the abstract API defined in this ETS to control the behaviour of an MHEG engine.

The following subclauses state requirements associated with the conformance of both implementations and applications to this ETS.

4.1 Implementation conformance

4.1.1 Conformance requirements

A conforming implementation for a language binding specification for this ETS shall meet all of the following criteria:

- the implementation shall support all required behaviour defined in this ETS;

- the implementation shall support all required interfaces defined in the language binding specification. Those interfaces shall support the behaviour described in this ETS and in the language binding specification;

- the implementation may provide additional functions or facilities not required by this ETS or by the language binding specification. Each such non-ETS extension shall be identified as such in the system documentation. Non-ETS extensions, when used, may change the behaviour of functions or facilities defined by this ETS or by the language binding specification. The conformance document shall define an environment in which an application can be run with the behaviour specified by this ETS and the language binding specification. In no case shall such an environment require modification of a Strictly Conforming Application.

4.1.2 Conformance documentation

A conformance document with the following information shall be available for an implementation claiming conformance to a language binding specification for this ETS. The conformance document shall be in two parts. The first part shall have the same structure as this ETS, with the information presented in the appropriately numbered sections, clauses, and subclauses. The second part shall have the same structure as the language binding specification, with the information presented in the appropriately numbered sections, clauses, and subclauses. The conformance document shall not contain information about extended features or capabilities outside the scope of this ETS and the language binding specification.

The conformance document shall identify the language binding specification to which the implementation conforms.

The conformance document shall contain a statement that indicates the full names, numbers, and dates of the language-independent and language binding specification ETSs that apply.

The conformance document shall state which of the optional features defined in this ETS and in the language binding specification are supported by the implementation.

The conformance document shall describe the behaviour of the implementation for all implementation-defined features defined in this ETS and in the language binding specification. This requirement shall be met by listing these features and by providing either a specific reference to the system documentation or full syntax and semantics of these features. The conformance document may specify the behaviour of the implementation for those features where this ETS or the language binding specification states that implementations may vary or where features are identified as undefined or unspecified.

No specifications other than those specified by this ETS and the language binding specification shall be present in the conformance document.

The phrases "shall document" or "shall be documented" in this ETS or in a language binding specification for this ETS mean that documentation of the feature shall appear in the conformance document, as described previously, unless the system documentation is explicitly mentioned.

The system documentation should also contain the information found in the conformance document.

4.2 Application conformance

All applications claiming conformance to a language binding specification for this ETS shall fall within one of the categories defined in the following subclauses.

4.2.1 Strictly conforming application

A strictly conforming application is an application that requires only the mandatory facilities described in this ETS, in the language binding specification and in the applicable language ETSs. Such an application shall accept a behaviour described in this ETS or in the language binding specification as unspecified or implementation defined and, for symbolic constants shall accept any value in the ranges permitted by this ETS and the language binding specification.

4.2.2 Conforming application

A Conforming Application of a language binding specification for this ETS is an application that differs from a Strictly Conforming Application in that it may use optional facilities described in this ETS, in the language binding specification and in the applicable language ETSs, as well as non-ETS facilities that are consistent with the ETS and with the language binding specification. Such an application shall fully document its requirements for these optional and extended facilities in addition to the documentation required of a Conforming Application.

4.3 Test methods

Any measurement of conformance to a language binding specification for this ETS shall be performed using test methods that conform to ISO/IEC 9646 Parts 1 to 5 [2] and to any additional requirements that may be imposed by the language binding specification.

5 General description

This clause situates the MHEG API within the architecture of an MHEG using application. It then specifies and classifies the software services that the MHEG API shall provide to its client applications.

5.1 Functional reference model of applications using MHEG

5.1.1 Reference model for multimedia applications

ETR 173 [3] and ETR 225 [4] define a generic reference model describing a functional architecture common to all multimedia applications making use of retrieval, conversational and/or distribution services. The model is applicable to all applications using MHEG.

Figures 1, 2 and 3 show how this reference model applies to applications using MHEG for terminal-to-host, terminal-to-terminal and terminal-to-database configurations in both point-to-point and multipoint communication.

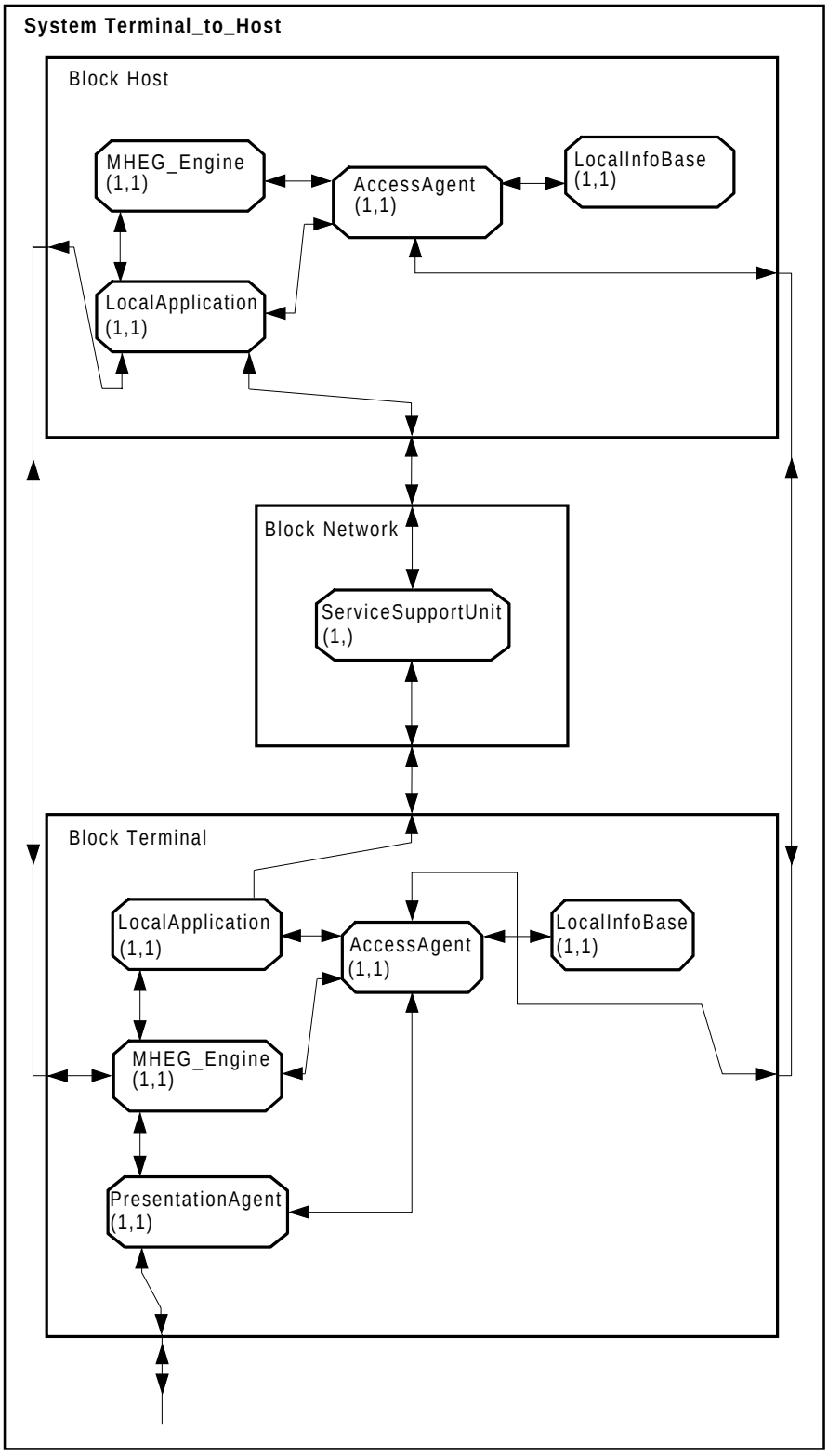


Figure 1: Application architecture reference model for terminal-to-host configurations

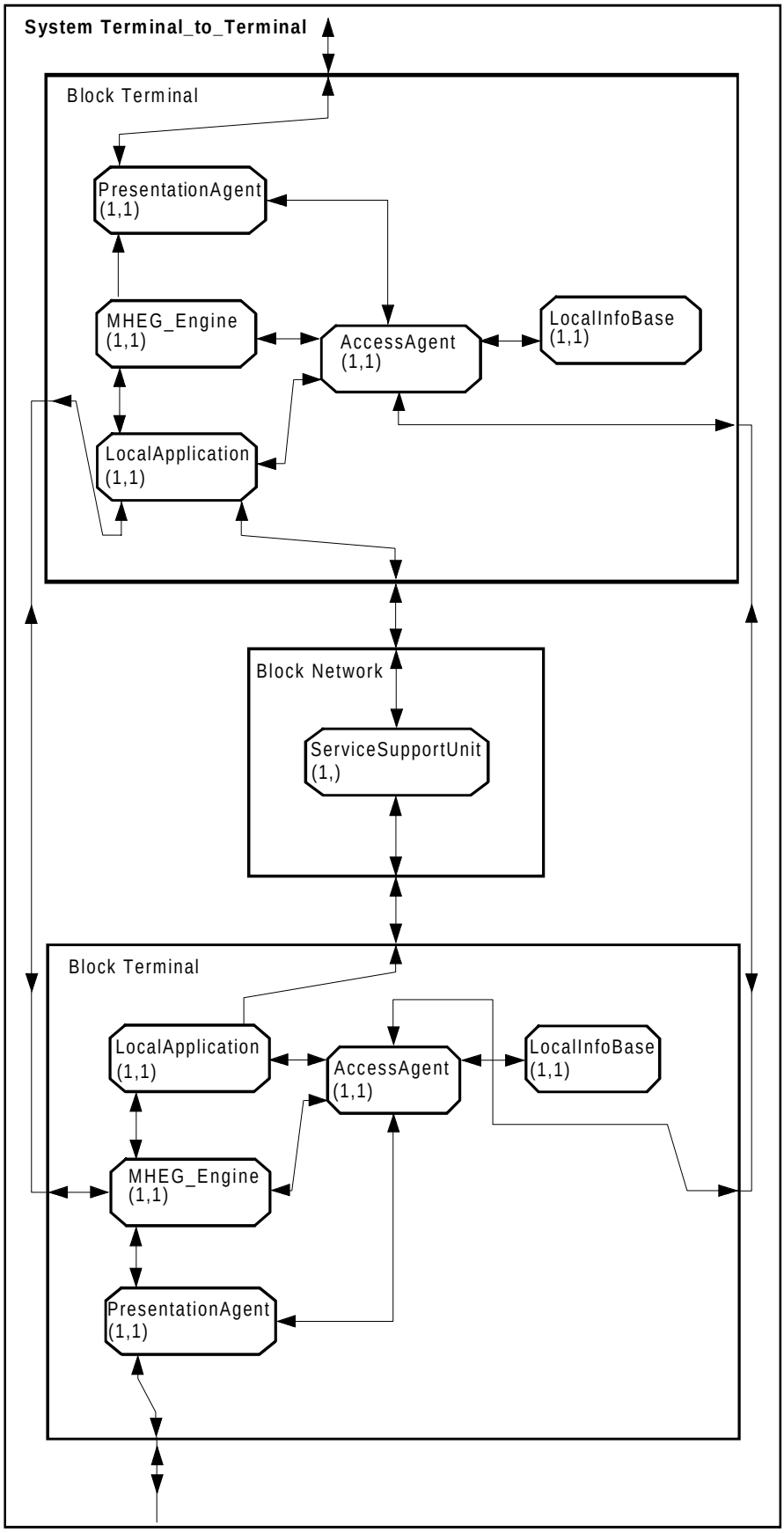


Figure 2: Application architecture reference model for terminal-to-terminal configurations

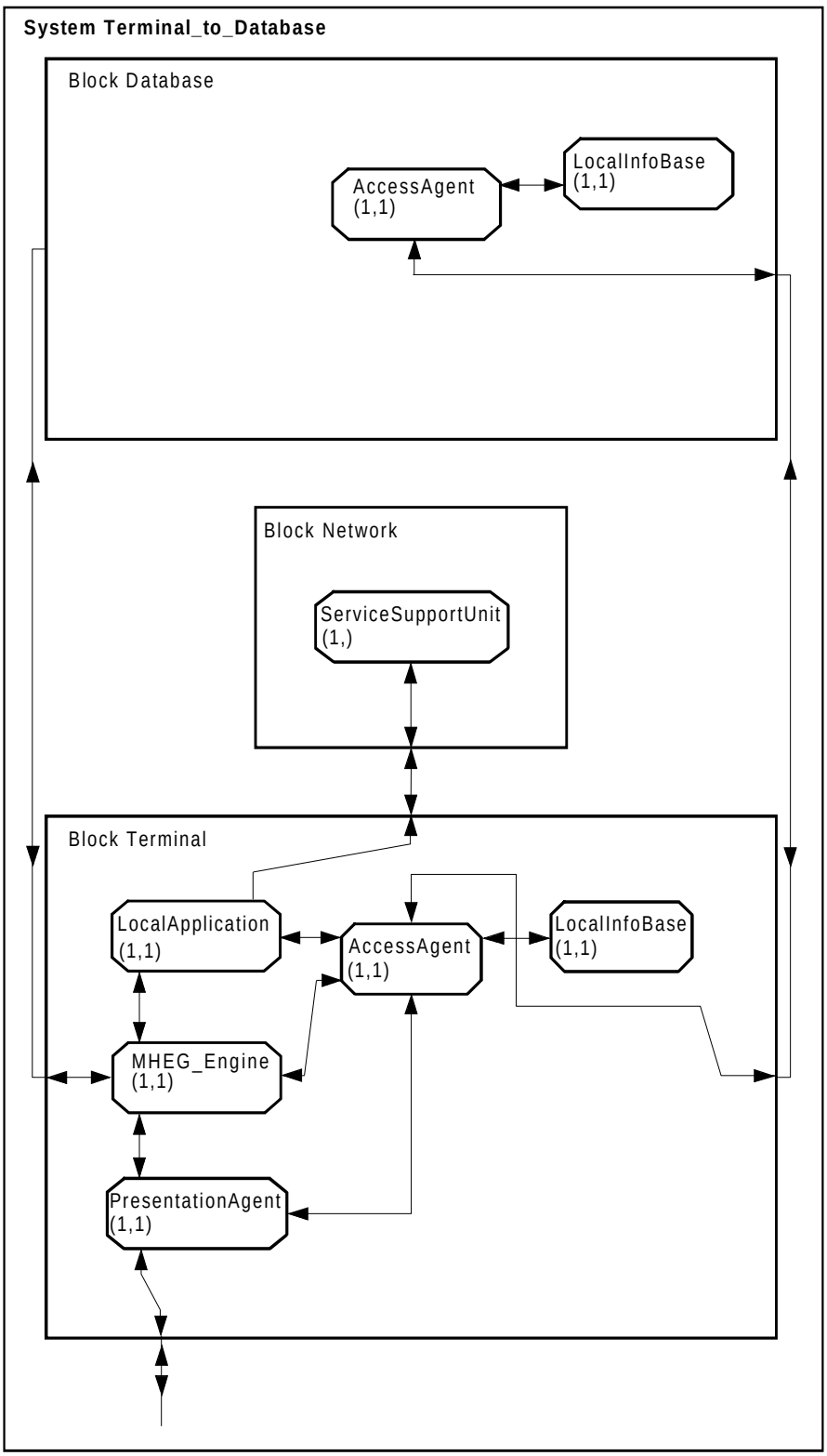


Figure 3: Application architecture reference model for terminal-to-database configurations

In a Multimedia and Hypermedia architecture the following functional units can be identified:

- presentation agent;
- access agent;
- local application interpreter;
- MHEG engine;
- local information base.

In a Multimedia and Hypermedia architecture the following Interfaces (APIs) can be identified:

- the services of the presentation agent are offered through the presentation API;
- the services of the MHEG engine are offered through the MHEG API;
- the services of the access agent are offered through the access API.

In a Multimedia and Hypermedia architecture the following end to end protocols can be identified:

- end-to-end protocol between an application - distant MHEG engine;
- end-to-end protocol between an application - service support function;
- end-to-end protocol between an access agent - access agent.

For a "terminal-to-host and database" configuration the host may use the end-to-end protocol between access agents to reference objects or contents at the database. The database has the same structure as shown in figure 3.

For a terminal-to-host architecture, the Service Support Unit (SSU) to which the terminal may be connected may also enable the user to select between different applications. The host is connected to the SSU via the host access network.

For a terminal-to-terminal architecture the SSU to which all terminals are connected may be a Multipoint Control Unit (MCU) that controls and manages the application and the different terminals. The end-to-end protocol between an application and a distant MHEG engine is applicable to all terminals. Each terminal application can use the protocol to communicate with any other terminal engine.

For a "terminal-to-database" or "terminal-to-host and database" structure the database may mainly consist of an access agent used to locate the referenced objects.

The presentation agent provides a multimedia content presentation service. It manages the presentation of monomedia data, performs data format decoding and manages the user interaction. It also acts as interface to external devices like smart card readers, VCRs, etc. The presentation agent is accessed by its clients (here the MHEG engine) through the "presentation API" which isolates the software on a higher level from the specific features of the various hardware sub-platforms. The presentation agent is only present on a terminal.

The access agent provides MHEG object and multimedia content location, access and communication services. It makes the location of the various objects (multimedia contents, MHEG objects, scripts, application specific data) transparent to its clients, i.e. the processes trying to access these objects. When an object is requested, the access agent is able first to locate it (possibly by issuing requests to directory services), then to retrieve it from local or distant storage (possibly by issuing requests to repository services and remote equipment), finally to provide access to it (possibly by using MHEG object and content encoding/decoding services). In the reverse way, the access agent also handles the forwarding of objects to remote equipment, their registration in directories and their storage in repositories.

The local application manages the logic of the application on a given platform. The application itself will often be distributed between several platforms (terminals, hosts). The local application is a client of the access agent via the access API, of the MHEG engine via the MHEG API and of the presentation agent via the presentation API. The local application may make use of a script execution service provided by a script processor. Scripts are parts of the application which are interchanged during the course of an application. A script processor is a functional unit able to execute scripts, it may be the script itself (if interchanged in executable form), a script language interpreter or an MHEG Script Interchange Representation (SIR) interpreter.

The MHEG engine provides an MHEG interpretation service. It interprets MHEG objects, manages links between them, triggers actions and orders objects presentation and access. It is controlled by the application through the MHEG API.

The local information base may be used to store objects, contents, scripts, etc. permanently or temporarily on the device. No assumption is made on how those objects are stored and which physical storage device is used.

The presentation API allows the terminal MHEG engine to access the multimedia content presentation service provided by a presentation agent.

The MHEG API allows MHEG applications to access the MHEG interpretation service provided by an MHEG engine. The MHEG API client application may be either the local application, or it may run on a remote device. If the application is running on a remote device it may access the MHEG API using the end-to-end protocol "application - distant MHEG engine" (CCITT Recommendation Q.9 [9]). The MHEG API is specified in this document.

The access API allows processes (e.g. local application, script interpreter, MHEG engine, presentation agent) to access the MHEG object and multimedia content location, access and communication services provided by an access agent.

The service support unit is a functional unit that handles service specific control parameters and offers functionality that depends on the particular service. An example for a service support unit is the MCU in the case of Videoconferencing service. The functionality handled by the service support unit is service specific.

The end-to-end protocol "application - distant MHEG engine" allows an application running on a distant device to communicate with a local MHEG engine. This protocol enables the implementation of terminal to host configurations.

The end-to-end protocol "application - service support unit" enables the use of the services provided by the service provider, by the terminal application or the user respectively.

The end-to-end protocol "access agent - access agent" enables the handling, maintenance and exchange of objects and data in a distributed environment.

5.1.2 The MHEG API

This subclause describes the terminology applicable to the MHEG API.

The MHEG API is the interface through which an MHEG application is allowed to control an MHEG engine.

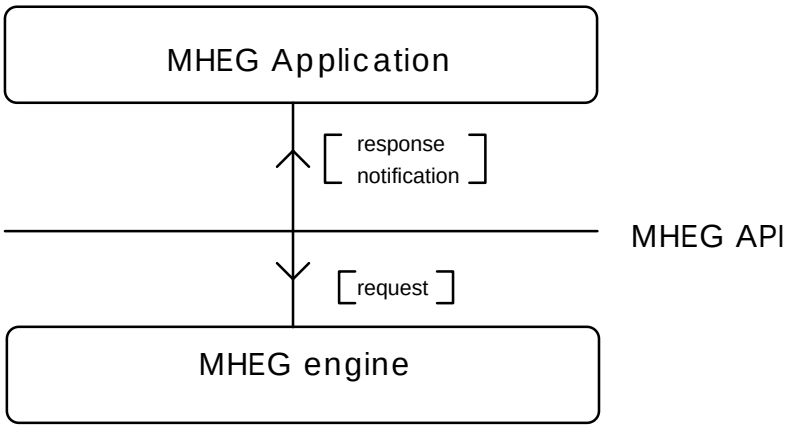


Figure 4: The MHEG API

An API consists of primitives, i.e. basic entry points provided by a provider module to any user module to enable the user to access software services supplied by the provider. These modules are pieces of software, although they can use services provided by computer hardware or other electronic equipment.

The MHEG API gives access to MHEG interpretation and (optionally) MHEG object access services. MHEG engines are the providers, whereas MHEG applications are the users. One MHEG engine may provide its software services to several applications. An MHEG engine can therefore be viewed as a server, whereas MHEG applications are the clients of the software services.

An API primitive is used to transfer some information between its user and its provider. This information consists of control and/or data. The information may be forwarded either from a client to its server or from the server to one of its clients. The information may be generated by its sender either on its own initiative or as a reply to a formerly issued primitive. The following terms are used:

- a request is a primitive issued by the client on its own initiative to forward information to the server;
- a response is a primitive issued by the server as a reply to a request to forward information to the client;
- a notification is a primitive issued by the server on its own initiative to forward information to the server.

Different kinds of requests may be considered:

- requests that require no response are called asynchronous requests;
- requests that do not require an immediate response are called deferred synchronous requests;
- requests that require an immediate response, until which the client process cannot proceed, are called synchronous requests.

Whether synchronous or asynchronous, requests may result in processing that will in turn trigger notifications.

5.2 Functional specification of the MHEG API

5.2.1 MHEG usage specifications

This subclause introduces MHEG usage rules. They consist of clarifications and interpretations of ISO/IEC DIS 13522-1 [1] regarding the definition of the main entities addressed by ISO/IEC DIS 13522-1 [1] that may be handled in applications using MHEG.

A clear understanding of the MHEG related entities implies a formal definition of the identity of the entities that are handled and/or transformed by components of an MHEG using application, as well of how they can be identified or referenced within different components.

5.2.1.1 Definitions

This subclauses introduces the concepts of identity, identification and referencing. The following precisions bring additional semantics with regard to ISO/IEC DIS 13522-1 [1] (which makes no clear distinction between reference and identification), and are applicable as definitions throughout this ETS.

The identity of an object is itself. The identity function is defined by Identity(A)=A. Any object has an identity. Object A and object B have the same identity if and only if A and B are the same object.

The identification of an object is an unambiguous way to determine its identity. If object A and object B have the same identifier, then they have the same identity, i.e. they are the same object. The identifier is often, but not necessarily, contained in the object. Objects may have several identification modes (with different types of identifier), though this is neither a useful nor recommended policy. If an object has no identifier, then it cannot be identified, although it has an identity.

Referencing an object is a convenient way to associate a name with an object. To determine the object which is referenced, the object reference needs to be resolved by some process. One reference shall only reference one object at a given time, but one object may be referenced in many ways. Unlike its identifier, references to an object are therefore not guaranteed to be unique. Moreover, it is possible to reference objects which otherwise are not identifiable.

5.2.1.2 MHEG objects

According to ISO/IEC DIS 13522-1 [1], an MHEG object is defined as a coded representation. Therefore, MHEG objects are bitstrings. The identity of an MHEG object is its bitstring. MHEG objects are "form a" objects as described in ISO/IEC DIS 13522-1 [1], subclause 6.2.4. MHEG object A and MHEG object B are identical if and only if they are the same sequence of bits.

An MHEG object is not a physical object, but rather an abstraction (a specified sequence of bits) which may have many representations (i.e. different objects) of different types: interchanged MHEG objects, stored MHEG objects, mh-objects, etc. Such representations are handled by different software services.

An MHEG object may be identified by an MHEG identifier. MHEG identifiers are the only way to identify MHEG objects. The structure and coded representation of MHEG identifiers is defined by ISO/IEC DIS 13522-1 [1]. The MHEG identifier of an MHEG object shall, if used, be encoded inside the MHEG object. Since the attribute is optional, some MHEG objects do not have an MHEG identifier. Such MHEG objects cannot be identified. The MHEG standard imposes a constraint on the design of MHEG using applications which is that MHEG object A and MHEG object B shall not have the same MHEG identifier unless they are identical.

The MHEG generic reference describes all possible ways to reference an MHEG object.

5.2.1.3 Mh-objects

An mh-object is an internal representation of an MHEG object within a process or system. An mh-object is not an MHEG object. Within an MHEG engine, mh-objects represent "available" MHEG objects. Mh-objects are "form b" objects as described in ISO/IEC DIS 13522-1 [1], subclause 6.2.4. An mh-object represents one MHEG object, i.e. there is always a bitstring that corresponds to an Mh-object. An MHEG engine shall not handle more than one mh-object to represent one MHEG object.

As a consequence, mh-objects handled by MHEG engines may be identified using MHEG identifiers. In addition, other mechanisms for identifying mh-objects (e.g. symbolic identification) may be defined by the application, provided their internal representation allows for it. This is especially useful when some of the MHEG objects represented by an MHEG engine's mh-objects are non-identifiable, i.e. have no MHEG identifier. This guarantees that all mh-objects shall be identifiable.

Mh-objects are referenced the same way as MHEG objects. References to MHEG objects for which the MHEG engine handles an mh-object will usually be resolved by addressing that mh-object.

5.2.1.4 Rt-objects

An rt-object is a run-time "instance" (or copy) of a "model" mh-object, which is created and handled by an MHEG engine for the purpose of presentation. An rt-object is not an MHEG object. Within an MHEG engine, rt-objects represent "rt-available" MHEG objects. Rt-objects are "form c" objects as described in ISO/IEC DIS 13522-1 [1], subclause 6.2.4. There may be none or several rt-objects which are "presentable" copies of one mh-object. An rt-object always has exactly one mh-object as its model.

Rt-objects may be identified using rt-object identifiers whose "model object identification" part is an MHEG identifier. The structure and coded representation of rt-object identifiers is defined by ISO/IEC DIS 13522-1 [1]. In addition, other mechanisms for identifying rt-objects (e.g. symbolic identification) may be defined by the application, provided their internal representation allows for it. This is especially useful when some of the MHEG objects represented by an MHEG engine's mh-objects used as models for rt-objects are non-identifiable, i.e. have no MHEG identifier. This guarantees that all rt-objects shall be identifiable.

Rt-objects may be referenced using MHEG generic references.

5.2.1.5 Channels

Channels are objects defined by ISO/IEC DIS 13522-1 [1] and handled by the MHEG engines. They may be identified using channel identifiers.

5.2.1.6 Interchanged MHEG objects

Interchanged MHEG objects are representations of MHEG objects which are being communicated at a given point in time using a network or storage medium. One given MHEG object (i.e. bitstring) may be interchanged many times between many places, i.e. represented by many interchanged MHEG objects. An MHEG external identifier may identify an interchanged MHEG object, and therefore reference an MHEG object through its location and time of interchange. However, it should be noted that an MHEG external identifier may not actually identify an MHEG object.

Stored MHEG objects are representations of MHEG objects which are usually located in files or database records. For example, one given MHEG object (i.e. bitstring) may be stored in many places, i.e. represented by many stored MHEG objects. Such locations are usually identified using file names or database identifiers. An MHEG external identifier may identify a storage location for an MHEG object, and therefore reference an MHEG object through its storage location.

5.2.2 Description of MHEG-related services

This subclause introduces the concept of MHEG-related services, i.e. common use software services handling MHEG-related entities. These services are provided by the building blocks (components) of the MHEG using application architecture upon which the MHEG application is built. Clients and services do not necessarily know each other and may be related through a mediating entity. In this building blocks approach, implementations of the services make use of each other.

An MHEG application may make use of some or all of these following MHEG-related services:

- the MHEG interpretation service allows the control of MHEG engine behaviour, i.e. the interpretation and presentation of MHEG objects;
- the MHEG object access service allows the access and modification of "logical" MHEG object's attributes;
- the MHEG object communication service allows the transfer of MHEG objects between locations;
- the MHEG object location service allows the management and resolution of references to MHEG objects;
- the MHEG object handling service allows the access management and interchange of physical MHEG objects;
- the MHEG object storage service allows the storage of MHEG object bitstrings and access to them;
- the MHEG object encoding/decoding service allows the encoding or decoding of MHEG objects;
- additional services for real-time distribution, presentation and production of MHEG objects and contents may be considered.

Different kinds of MHEG related entities, bearing strong relationships with each other but being nevertheless of different types (therefore not comparable), are being handled by these software services:

- the MHEG interpretation service handles rt-objects, sockets, channels and mh-objects;
- the MHEG object access service handles mh-objects;
- the MHEG object handling service handles MHEG objects. Since MHEG objects are virtual rather than physical objects, this service relies on services that handle physical "representations" of MHEG objects, such as storage or transport services;
- the MHEG object location service handles MHEG generic references. Through the use of maps, it is able to resolve such references, i.e. translate them into identifiers understandable by the requesting application;
- the MHEG object communication service handles interchanged MHEG objects;
- the MHEG object storage service handles stored MHEG objects and manages MHEG object storage locations, whose type depend on the underlying storage mechanism, e.g. files, database records;
- the MHEG object encoding/decoding service provides functions for transforming mh-objects into MHEG objects and vice versa.

The MHEG API consists of the interface provided by the following services, that shall be provided by conforming MHEG engines:

- the MHEG interpretation service (mandatory);
- the MHEG object access service (optional).

6 API definition principles

6.1 Satisfaction of technical requirements on the MHEG API

The MHEG API is defined as an abstract API specification, i.e. a language-independent description of the semantics of a set of functionality in an abstract syntax using abstract data types.

Following the recommendations of ETR 225 [4], the MHEG API ETS should meet the following requirements:

- be portable;
- be generic;
- be conformance testable;
- be implementable.

The **portability** requirement states that the MHEG API ETS shall enable MHEG applications to use the MHEG object manipulation and interchange service provided by MHEG engines independently of:

- the programming language used for the MHEG application;
- the underlying operating system.

This ETS meets the portability requirement by the definition of an abstract API specification.

The being **generic** requirement states that the MHEG API ETS shall provide appropriate support to cover all the common requirements of MHEG applications.

This ETS meets the being generic requirement through defining the MHEG API at the most basic level, e.g. by defining primitives that match MHEG elementary actions and data types that match MHEG data types. This guarantees maximisation of the range of MHEG object manipulations made available to applications.

The **conformance testability** requirement states that the MHEG API ETS should make it as easy as possible to ensure the conformance of MHEG engines to the MHEG API ETS, i.e. the correct provision of this API by an MHEG engine under test as well as the conformance of MHEG applications to the MHEG API ETS, i.e. the correct use of this API by an MHEG application under test.

This ETS meets the conformance testability requirement by formal expression of the requirements on conforming implementations and conforming applications, as well as by the use of a formal description technique for the definition of the MHEG API.

The **implementability** requirement states that the MHEG API ETS should take into account simplicity and clarity both in the definition and the formulation to make implementation of conforming MHEG engines as easy as possible.

This ETS meets the implementability requirement by the provision of informative guidelines to deduce language binding specifications and message encoding rules from the abstract API specification.

6.2 Use of Interface Definition Language

The MHEG API is defined using Interface Definition Language (IDL) ISO/IEC 14750-1 [10].

6.2.1 Comprehensive introduction to IDL

IDL is a formal description technique for specifying the services provided by objects for use by applications or other objects. Although object-oriented communication in distributed environments is actually a technology of some relevance with regard to the definition of a multimedia core toolbox, this ETS only considers the use of IDL and its underlying object model as a context-independent formal description technique for the specification of APIs.

Application of IDL should be based on an underlying object model. Such an object model is defined in terms of **object types** which support **operations** characterising the behaviour of objects. **Objects** are instances of object types. Objects may be identified using **object references**. **Non-object types** can be instantiated but do not support operations. Operations are defined by a **signature** consisting of a name, a list of input or output **parameter** types and a list of **result** types. The set of operation signatures defined for a type is the **interface** of that type. **Subtyping** allows the definition of type hierarchies, with subtypes providing their supertype's interface as a part of their own interface. **Operation requests** may have different operational semantics such as synchronous, asynchronous, etc. Consequences of an operation request include side effects, results and **exceptions**.

IDL is the language used to describe the interfaces (i.e. the set of operations) provided by objects. It consists of lexical conventions, preprocessing directives and an Extended Backus Naur Form (EBNF) grammar. An IDL specification of an API consists of data type definitions, constant definitions, exception definitions, interface definitions and module definitions.

6.2.2 The Interface Definition Language (IDL)

This subclause describes the main concepts that are necessary to understand the MHEG API definition.

The object model provides an organised representation of objects concepts and terminology. It defines a partial model for computation that embodies the key characteristics of objects as realised by the presented technologies.

The object model presented in this ETS is abstract in that is not directly realised by any particular technology.

An objects system provides services to its clients. A client of a service is any entity capable of requesting the service.

6.2.2.1 Objects

An object system includes entities known as objects. An object is an identifiable, encapsulated entity that provides one or more services that can be requested by a client.

6.2.2.2 Requests

Clients request services by issuing requests. A request is an event, i.e. something that occurs at a particular time. The information associated with a request consists of an operation, a target object, zero or more (actual) parameters, and an optional request context.

A **request form** is a description or pattern that can be evaluated or performed multiple times to cause the issuing of requests.

A **value** is anything that may be a legitimate (actual) parameter in a request. A value may identify an object, for the purpose of performing the request. A value that identifies an object is called object name.

An **object reference** that reliably denotes a particular object. Specifically, an object reference will identify the same object each time the reference is used in a request (subject to certain pragmatic limit in space and time). An object may be denoted by multiple, distinct object references.

A request may have parameters that are used to pass data to the target object; it may also have a request context which provides additional information about the request.

A request causes a service to be performed on behalf of the client. One outcome of performing a service is returning to the client the results, if any, defined for the request.

If an abnormal condition occurs during the performance of a request, an **exception** is returned. The exception may carry additional return parameters particular to the exception.

The request parameters are identified by position. A parameter may be an input parameter, an output parameter, or an input-output parameter. A request may also return a single result value, as well as any output parameters.

The following semantics apply for all requests:

- any aliasing of parameter values is neither guaranteed removed nor guaranteed preserved;
- the order in which aliased output parameters are written is not guaranteed;
- any output parameters are undefined if an exception is returned;
- the values which may be returned in an input-output parameter may be constrained by the value which was input.

6.2.2.3 Types

A type is an identifiable with an associated predicate (a single-argument mathematical function with a boolean result) defined over values. A value satisfies a type if the predicate is true for that value. A value that satisfies a type is called **member of the type**.

Types are used in signatures to restrict a possible parameter or to characterise a possible result.

The **extension** of the type is the set of values that satisfy the type at any particular time.

An object type is a type whose members are objects (literally, values that identify objects). In other words, an object type is satisfied only by (values that identify) objects.

6.2.2.4 Interfaces

An **interface** is a description of a set of possible operations that a client may request of an object. An object satisfies an interface if it can be specified as target object in each potential request described by the interface.

An **interface type** is a type that is satisfied by any object (literally, any value that identifies an object) that satisfies a particular interface.

6.2.2.5 Operations

An **operation** is an identifiable entity that denotes a service that can be requested.

An operation is identified by an **operation identifier**. An operation is not a value.

An operation has a signature that describes the legitimate values of request parameters and returned results. In particular a signature consists of:

- a specification of parameters required in requests for that operation;
- a specification of the results of the operations;
- a specification of the exceptions that may be raised by a request for the operation and the types of parameters accompanying them;
- a specification of additional contextual information that may affect the request;
- an indication of the execution semantics the client should expect from a request for the operation.

A **parameter** is characterised by its mode and its type. The mode indicates whether the value should be passed from the client to the server (in), from the server to the client (out), or both (inout). The parameter's type constrains the possible value which may be passed in the direction[s] dictated by the mode.

The **return result** is a distinguished out parameter.

An **exception** is an indication that an operation request was not performed successfully. An exception may be accompanied by additional exception-specific information.

A **request context** provides additional, operation-specific information that may effect the performance of a request.

Two styles of **execution semantics** are defined by the object model:

- at-most-once: if an operation request returns successfully, it was performed exactly once; if it returns an exception indication, it was performed at most once;
- best-effort: a best-effort operation is a request-only operation, i.e. it cannot return any results and the requester never synchronises with the completion, if any, of the request.

The execution semantics to be expected are associated with an operation. This prevents a client and object implementation from assuming different execution semantics.

Note that the client is able to invoke an at-most-once operation in a synchronous or deferred-synchronous manner.

6.2.2.6 Attributes

An interface may have attributes. An attribute is logically equivalent to declaring a pair of accessor functions: one to retrieve the value of the attribute and one to set the value of the attribute.

An attribute may be read-only, in which case only the retrieval accessor function is defined.

6.2.2.7 Subtyping versus inheritance

Subtyping is a relationship between types based on their interfaces. It defines the rules by which objects of one type are determined to be acceptable in contexts expecting another type. Inheritance is a mechanism for reuse. Many object systems do not distinguish between subtyping and inheritance. The following subclause defines the two concepts separately, but then explicitly states how they are related.

6.2.2.8 Subtyping

The object model supports subtyping for object types. Intuitively, a type is a subtype of another if the first is a specialisation or refinement of the second. Operationally it means that any object of the first type can be used in any context that expects an object of the second type; that is, if S is a subtype of T, an object of type S may be used wherever an object of type T may be used. In other words, objects of type S are also of type T. Subtypes can have multiple parent types, with the implication that an object that is an instance of type S is also an instance of all supertypes of type S. The relationships between types define a type hierarchy, which can be drawn as a directed acyclic graph.

6.2.2.9 Inheritance

Inheritance is a notational mechanism for defining a type S in terms of another type T. The definition of S inherits all the operations of T and may provide other operations. Intuitively, inherits means that the operations defined for T are also defined for or can be used by S. Subtyping is a relationship between interfaces (types). Inheritance can apply to both interfaces and implementations; that is both interfaces and implementations can be inherited. The object model is concerned with inheritance of interfaces. It does not specify what can happen with implementations of inherited operations (for example, whether they may be changed or overridden by a subtype).

6.2.3 Principles for mapping IDL interfaces to API primitives

The MHEG engine interface consists of a set of API primitives that can be organised into clusters according to the target entity of a primitive. Definition of this interface can therefore logically be structured according to the operation provider. In the MHEG API context, objects that provide interfaces need not be implemented as separate object implementations. More likely, they would be internal entities handled by the MHEG engine. As for ISO/IEC DIS 13522-1 [1], the MHEG API definition follows an object-oriented methodology without requiring the implementations to use object-oriented design or programming techniques.

Services are defined not as the interface that a module should provide but as a set of operations conspiring to offer a service, to which clients should have access.

The MHEG API therefore consists of IDL interface objects which provide operations that map API primitives. The object instance on which an operation is requested corresponds to the main (target) parameter of the API primitive.

6.2.4 Fulfilment of technical requirements

The use of IDL contributes to the fulfilment of the portability and implementability technical requirements:

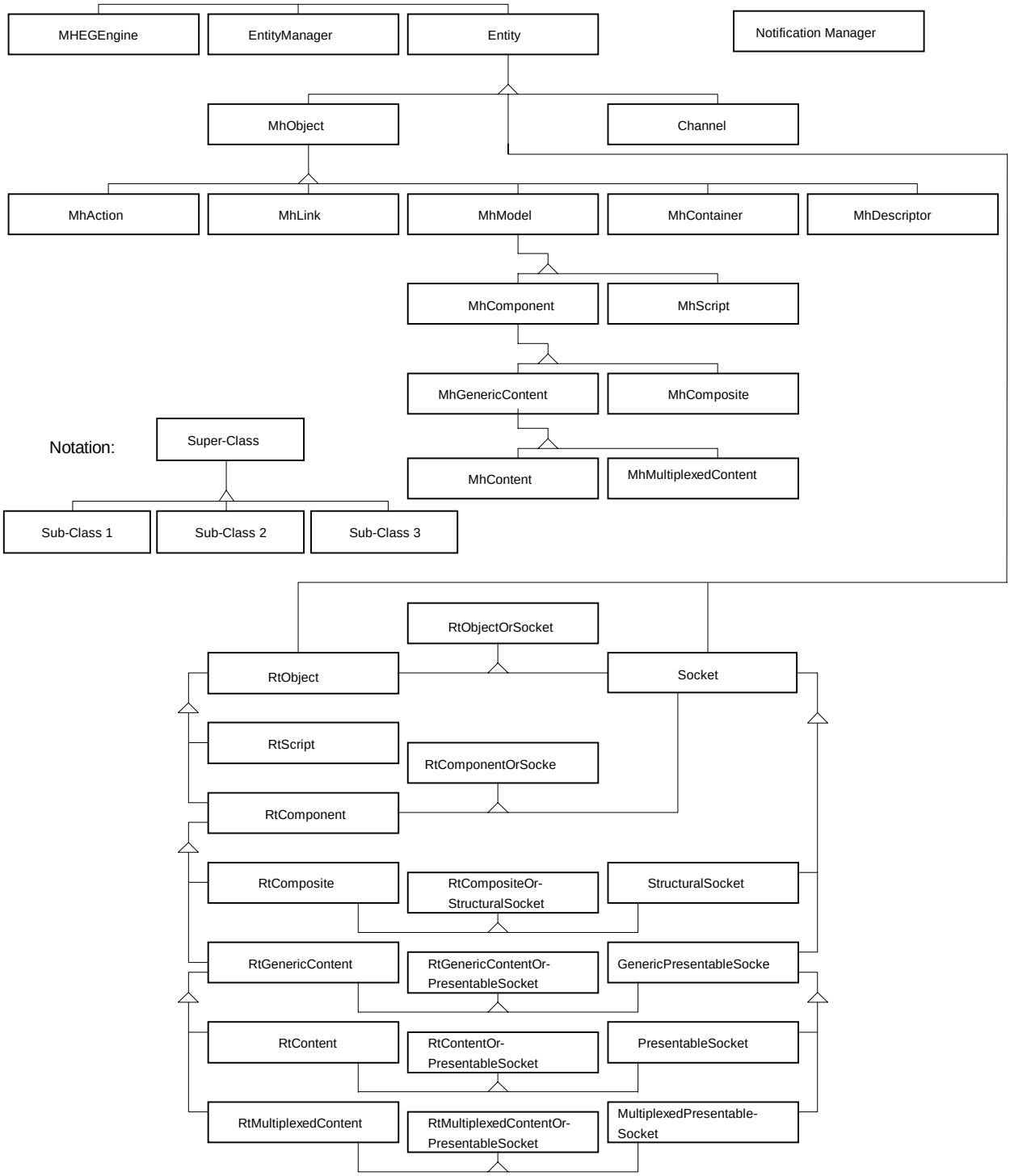
- IDL is independent from a programming language. Moreover publicly available specifications for IDL language binding to C and C++ exist, and others are under study;
- IDL provides a complete formal description language which allows a very concise, readable and efficient specification of the MHEG API. Moreover, this formal description language is also appropriate for automatic compilation, which means that MHEG API implementations could be automatically generated for a given language and operating system using appropriate IDL compilers. This of course could be a major element in facilitating implementability and general use of the ETS API rather than any specific interface.

6.3 Overview of the API definition and general principles

6.3.1 The MHEG API object model

This subclause presents the object model, i.e. the object types (interfaces) provided by the MHEG API and their subtyping relationships.

It may be noted that the objects described hereafter are introduced as useful concepts for specifying the interface, but are not required to be implemented as separate objects. The MHEG API is specified as an abstract API in terms of operations provided by objects, but implementations of the MHEG API will be provided by MHEG engine implementations.



NOTE 1: MhObjects (and their subtypes) match form b) objects as defined in ISO/IEC DIS 13522-1 [1], subclause 6.2.4, i.e. objects available to the MHEG engine.

NOTE 2: RtObjects (and their subtypes) match form c) objects as defined in ISO/IEC DIS 13522-1 [1], subclause 6.2.4, i.e. instances of MhObjects available to the presentation process.

Figure 5: Object model

7 Definition of the MHEG API

7.1 Mandatory primitives

7.1.1 MHEGEngine object

The following subclause defines the operations of the `MHEGEngine` object.

7.1.1.1 `initialiseEngine` operation

Synopsis:

Interface: `MHEGEngine`
Operation: `initialiseEngine`
Result: `void`

Description:

This operation performs any necessary initialisation of the interface. It shall be invoked before any other interface operations defined in this ETS are invoked. It can be invoked multiple times, in which case each invocation shall reinitialise the MHEG Engine.

7.1.1.2 `shutdownEngine` operation

Synopsis:

Interface: `MHEGEngine`
Operation: `shutdownEngine`
Result: `void`

Description:

This operation deletes all service-generated interface objects associated with the current session. The next operation that shall be accepted by an MHEG engine is an `initialiseEngine` operation.

7.1.1.3 IDL description

```
interface MHEGEngine    {

    void    initialiseEngine();
    void    shutdownEngine();

};
```

7.1.2 NotificationManager object

The following subclause defines the operations of the `NotificationManager` object.

7.1.2.1 `getReturnability` operation

Synopsis:

Interface: `NotificationManager`
Operation: `getReturnability`
Result: `sequence<unsigned short>`

Description:

This operation retrieves the returnability behaviour of the MHEG engine.

The operation returns a list of numbers identifying available notifications.

7.1.2.2 `getNotification` operation

Synopsis:

Interface:	`NotificationManager`	
Operation:	`getNotification`	
Result:	`void`	
In:	`unsigned short`	`notification_number`
Out:	`sequence<GenericValue>`	`values`
Out:	`sequence<MhObjectReference>`	`objects`
Exception:	`InvalidParameter`	

Description:

This operation retrieves a notification from the MHEG engine.

The `notification_number` parameter identifies the notification. This identification may be the result of a `getReturnability` operation.

The `values` parameter specifies the returned values.

The `objects` parameter specifies the returned object references.

The `InvalidParameter` exception is raised when the value of one of the parameters prohibits the normal execution of the action. The `completion_status` member indicates whether the action was completed (with a default value assigned to the inadequate parameter) or not. The `parameter_number` member identifies the rank of the invalid parameter.

7.1.2.3 IDL description

```
interface   NotificationManager {

    sequence<unsigned short>
        getReturnability();

    void
        getNotification(
            in unsigned short
                notification_number,
            out sequence<GenericValue>
                values,
            out sequence<MhObjectReference>
                objects)
    raises(InvalidParameter);

};
```

7.1.3 `EntityManager` object

The following subclause defines the operations of the `EntityManager` object.

7.1.3.1 `getAvailableMhObjects` operation

Synopsis:

Interface:	`EntityManager`
Operation:	`getAvailableMhObjects`
Result:	`sequence<MHEGIdentifier>`

Description:

This operation retrieves the mh-objects available to the MHEG engine.

An mh-object is either "not ready" (in period O1), "processing" (in period O2 or O4) or "ready" (in period O3). The operation retrieves those mh-objects which are in period O3.

The operation returns the identifiers of the available mh-objects.

7.1.3.2 `getAvailableRtObjects` operation

Synopsis:

```
Interface:    EntityManager
Operation:    getAvailableRtObjects
Result:       sequence<RtObjectIdentifier>
```

Description:

This operation retrieves the rt-objects available to the MHEG engine.

An rt-object is either "not available" (in period R1), "processing" (in period R2 or R4) or "available" (in period R3 and its subperiods). The operation retrieves those rt-objects which are in period R3.

The operation returns the identifiers of the available rt-objects.

7.1.3.3 `getAvailableChannels` operation

Synopsis:

```
Interface:    EntityManager
Operation:    getAvailableChannels
Result:       sequence<ChannelIdentifier>
```

Description:

This operation retrieves the channels available to the MHEG engine.

A channel is either "non available" (in period C1), "processing" (in period C2 or C4) or "available" (in period C3). The operation retrieves those channels which are in period C3.

The operation returns the identifiers of the available channels.

7.1.3.4 `releaseAlias` operation

Synopsis:

```
Interface:    EntityManager
Operation:    releaseAlias
Result:       void
In:           string                          alias
Exception:    InvalidParameter
```

Description:

This operation enables to release an alias. It cancels the assignments of this alias to entities.

The `alias` parameter specifies the value of the released alias.

The `InvalidParameter` exception is raised when the value of one of the parameters prohibits the normal execution of the action. The `completion_status` member indicates whether the action was completed (with a default value assigned to the inadequate parameter) or not. The `parameter_number` member identifies the rank of the invalid parameter.

7.1.3.5 IDL description

```
interface   EntityManager   {

    sequence<MHEGIdentifier>
        getAvailableMhObjects();

    sequence<RtObjectIdentifier>
        getAvailableRtObjects();

    sequence<ChannelIdentifier>
        getAvailableChannels();

    void
        releaseAlias(
            in string
                alias)
    raises(InvalidParameter);
};
```

7.1.4 `Entity` object

The following subclause defines the operations of the `Entity` object.

7.1.4.1 `setAlias` operation

Synopsis:

Interface:	`Entity`	
Operation:	`setAlias`	
Result:	`void`	
In:	`string`	`alias`
Exception:	`InvalidTarget`	

Description:

This operation enables the assignment of an alias to any entity.

The `setAlias` operation triggers the execution of the "set alias" elementary action with the bound entity as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 34.2.1.

The `alias` parameter specifies the value of the "alias" parameter of the "set alias" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.4.2 `getAlias` operation

Synopsis:

Interface:	`Entity`
Operation:	`getAlias`
Result:	`string`
Exception:	`InvalidTarget`

Description:

This operation retrieves the alias assigned to an entity.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.4.3 IDL description

```
interface Entity    {

    void
        setAlias(
            in string
                alias)
    raises(InvalidTarget);

    string
        getAlias()
    raises(InvalidTarget);
};
```

7.1.5 `MhObject` object

The following subclause defines the operations of the `MhObject` object. The object inherits from the `Entity` object.

7.1.5.1 `bind` operation

Synopsis:

Interface:	`MhObject`	
Operation:	`bind`	
Result:	`MHEGIdentifier`	
In:	`MhObjectReference`	`mh_object_reference`
Exception:	`AlreadyBound`	
Exception:	`InvalidTarget`	

Description:

This operation binds the MhObject instance (an interface object instance) with an MHEG object (an MHEG entity).

The `mh_object_reference` parameter specifies the reference of the MHEG object.

The operation returns the identifier of the bound MHEG object.

The `AlreadyBound` exception is raised when the interface object instance is already bound with an MHEG entity.

The `InvalidTarget` exception is raised when the targeted MHEG entity is not available. The `period` member returns the current period of the target.

7.1.5.2 `unbind` operation

Synopsis:

Interface:	`MhObject`
Operation:	`unbind`
Result:	`void`
Exception:	`NotBound`

Description:

This operation cancels the binding between the MhObject instance (an interface object instance) and an MHEG object (an MHEG entity).

The `NotBound` exception is raised when the interface object instance is not bound with an MHEG entity.

7.1.5.3 `prepare` operation

Synopsis:

Interface:	`MhObject`	
Operation:	`prepare`	
Result:	`MHEGIdentifier`	
In:	`MhObjectReference`	`mh_object_reference`
Exception:	`AlreadyBound`	
Exception:	`InvalidTarget`	

Description:

This operation enables the creation of an MHEG object from a model object by the MHEG engine.

The `prepare` operation triggers the execution of the "prepare" elementary action targeted at a single MHEG object.

The effect of the action on its target and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 36.2.1.

The `mh_object_reference` parameter specifies a reference to an MHEG object.

This operation implicitly binds the MhObject instance (an interface object instance) with the new prepared MHEG object (an MHEG entity).

The operation returns the identifier of the new prepared MHEG object bound with the MhObject instance.

The `AlreadyBound` exception is raised when the interface object instance is already bound with an MHEG entity.

The `InvalidTarget` exception is raised when the targeted MHEG entity is not available. The `period` member returns the current period of the target.

7.1.5.4 `destroy` operation

Synopsis:

Interface:	`MhObject`
Operation:	`destroy`
Result:	`void`
Exception:	`NotBound`
Exception:	`InvalidTarget`

Description:

This operation enables the removing of an MHEG object by the MHEG engine.

The `destroy` operation triggers the execution of the "destroy" elementary action targeted at a single MHEG object.

The effect of the action on its target and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 36.2.2.

This operation implicitly cancels the binding between the MhObject instance (an interface object instance) and the new destroyed MHEG object (an MHEG entity).

The `NotBound` exception is raised when the interface object instance is not bound with an MHEG entity.

The `InvalidTarget` exception is raised when the targeted MHEG entity is not available. The `period` member returns the current period of the target.

7.1.5.5 `getPreparationStatus` operation

Synopsis:

Interface: `MhObject`
Operation: `getPreparationStatus`
Result: `PreparationStatusValue`
Exception: `NotBound`
Exception: `InvalidTarget`

Description:

This operation retrieves the availability of an MHEG object to the MHEG engine.

The `getPreparationStatus` operation triggers the execution of the "get preparation status" elementary action with the bound MHEG object as its single target.

The effect of the action on its target, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 36.3.1.

The operation returns the availability of the MHEG object bound with the MhObject instance. The returned value is either `NOT_READY`, `PROCESSING` or `READY`.

When the returned value is `NOT_READY`, the operation implicitly cancels the binding between the MhObject instance (an interface object instance) and the MHEG object (an MHEG entity).

The `NotBound` exception is raised when the interface object instance is not bound with an MHEG entity.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.5.6 `getIdentifier` operation

Synopsis:

Interface: `MhObject`
Operation: `getIdentifier`
Result: `MHEGIdentifier`
Exception: `NotBound`

Description:

This operation retrieves the identifier of the MHEG object (an MHEG entity) bound with the MhObject instance (an interface object instance).

The `NotBound` exception is raised when the interface object instance is not bound with an MHEG entity.

7.1.5.7 `kill` operation

Synopsis:

Interface: `MhObject`
Operation: `kill`
Result: `void`

Description:

This operation deletes the MhObject instance (an interface object instance).

7.1.5.8 IDL description

```
interface   MhObject: Entity    {

    MHEGIdentifier
        bind(
            in MhObjectReference
                mh_object_reference)
    raises(AlreadyBound, InvalidTarget);

    void
        unbind()

    raises(NotBound);

    MHEGIdentifier
        prepare(
            in MhObjectReference
                mh_object_reference)
    raises(AlreadyBound, InvalidTarget);

    void
        destroy()
    raises(NotBound, InvalidTarget);

    PreparationStatusValue
        getPreparationStatus()
    raises(NotBound, InvalidTarget);

    MHEGIdentifier
        getIdentifier()
    raises(NotBound);

    void
        kill();

};
```

7.1.6 `MhAction` object

The following subclause defines the operations of the `MhAction` object. The object inherits from the `MhObject` object.

7.1.6.1 `delay` operation

Synopsis:

Interface:	`MhAction`	
Operation:	`delay`	
Result:	`void`	
In:	`unsigned short`	`nested_action_number`
In:	`unsigned long`	`delay`
Exception:	`InvalidTarget`	
Exception:	`InvalidParameter`	

Description:

This operation enables to delay the process of nested actions within the mh-action.

The `nested_action_number` parameter specifies the nested action after which the delay is to be processed.

The `delay` parameter specifies the duration of the delay expressed in Generic Temporal Unit (GTU).

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

The `InvalidParameter` exception is raised when the value of one of the parameters prohibits the normal execution of the action. The `completion_status` member indicates whether the action was completed (with a default value assigned to the inadequate parameter) or not. The `parameter_number` member identifies the rank of the invalid parameter.

7.1.6.2 IDL description

```
interface   MhAction: MhObject  {

    void
        delay(
            in unsigned short
                nested_action_number,
            in unsigned long
                delay)
    raises(InvalidTarget, InvalidParameter);

};
```

7.1.7 `MhLink` object

The following subclause defines the operations of the `MhLink` object. The object inherits from the `MhObject` object.

7.1.7.1 `abort` operation

Synopsis:

Interface: `MhLink`
Operation: `abort`
Result: `void`
Exception: `InvalidTarget`

Description:

This operation aborts the processing of all the actions that have been activated by a link object. Each time the link condition is satisfied, the actions defining the link effect are activated and processed.

The `abort` operation triggers the execution of the "abort" elementary action with the bound link object as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 38.2.1.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.7.2 IDL description

```
interface   MhLink: MhObject    {

    void
        abort()
    raises(InvalidTarget);
};
```

7.1.8 `MhModel` object

For the `MhModel` object no specific operations are defined. The object inherits from the `MhObject` object.

7.1.8.1 IDL description

```
interface   MhModel: MhObject {};
```

7.1.9 `MhComponent` object

For the `MhComponent` object no specific operations are defined. The object inherits from the `MhModel` object.

7.1.9.1 IDL description

```
interface   MhComponent: MhModel {};
```

7.1.10 `MhGenericContent` object

The following subclause defines the operations of the `MhGenericContent` object. The object inherits from the `MhComponent` object.

7.1.10.1 `copy` operation

Synopsis:

Interface:	`MhGenericContent`	
Operation:	`copy`	
Result:	`void`	
In:	`sequence<MhObjectReference>`	`copies`
Exception:	`InvalidTarget`	
Exception:	`InvalidParameter`	

Description:

This operation specifies the copy of a content object "source" in a set of content objects "copies" or the copy of a multiplexed content object "source" in a set of multiplexed content objects "copies".

The `copy` operation triggers the execution of the "copy" elementary action with the bound content object or multiplexed content object as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 40.2.1.

The `copies` parameter specifies the value of the "copies" parameter of the "copy" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

The `InvalidParameter` exception is raised when the value of one of the parameters prohibits the normal execution of the action. The `completion_status` member indicates whether the action was completed (with a default value assigned to the inadequate parameter) or not. The `parameter_number` member identifies the rank of the invalid parameter.

7.1.10.2 IDL description

```
interface   MhGenericContent: MhComponent   {

    void
        copy(
            in sequence<MhObjectReference>
                copies)
    raises(InvalidTarget, InvalidParameter);
};
```

7.1.11 `MhContent` object

The following subclause defines the operations of the `MhContent` object. The object inherits from the `MhGenericContent` object.

7.1.11.1 `setData` operation

Synopsis:

Interface:	`MhContent`	
Operation:	`setData`	
Result:	`void`	
In:	`boolean`	`substitution_indicator`
In:	`sequence<DataElement>`	`data_elements`
Exception:	`InvalidTarget`	
Exception:	`InvalidParameter`	

Description:

This operation allows to store or to modify the generic value in the data of a content object.

The `setData` operation triggers the execution of the "set data" elementary action with the bound content object as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 39.2.1.

The `substitution_indicator` parameter specifies the value of the "substitution indicator" parameter of the "set data" action.

The `data_elements` parameter specifies the value of the "data elements" parameter of the "set data" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

The `InvalidParameter` exception is raised when the value of one of the parameters prohibits the normal execution of the action. The `completion_status` member indicates whether the action was completed (with a default value assigned to the inadequate parameter) or not. The `parameter_number` member identifies the rank of the invalid parameter.

7.1.11.2 `getData` operation

Synopsis:

Interface:	`MhContent`	
Operation:	`getData`	
Result:	`GenericValue`	
In:	`sequence<long>`	`element_list_index`
Exception:	`InvalidTarget`	
Exception:	`InvalidParameter`	

Description:

This operation retrieves a generic value or an element of a generic list stored in the data of a content object.

The `getData` operation triggers the execution of the "get data" elementary action with the bound content object as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 39.3.1.

The `element_list_index` parameter specifies the value of the "element list index parameter" parameter of the "get data" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

The `InvalidParameter` exception is raised when the value of one of the parameters prohibits the normal execution of the action. The `completion_status` member indicates whether the action was completed (with a default value assigned to the inadequate parameter) or not. The `parameter_number` member identifies the rank of the invalid parameter.

7.1.11.3 IDL description

```
interface   MhContent: MhGenericContent {

    void
        setData(
            in boolean
                substitution_indicator,
            in sequence<DataElement>
                data_elements)
    raises(InvalidTarget, InvalidParameter);

    GenericValue
        getData(
            in sequence<long>
                element_list_index)
    raises(InvalidTarget, InvalidParameter);

};
```

7.1.12 `MhMultiplexedContent` object

The following subclause defines the operations of the `MhMultiplexedContent` object. The object inherits from the `MhGenericContent` object.

7.1.12.1 `setMultiplex` operation

Synopsis:

Interface:	`MhMultiplexedContent`	
Operation:	`setMultiplex`	
Result:	`void`	
In:	`sequence<StreamIdentifier>`	`stream_list`
Exception:	`InvalidTarget`	
Exception:	`InvalidParameter`	

Description:

This operation specifies the multiplexing of a list of content objects, the result is set in one multiplexed content object containing the multiplexed data.

The `setMultiplex` operation triggers the execution of the "set multiplex" elementary action with the bound multiplexed content object as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 41.2.1.

The `stream_list` parameter specifies the value of the "stream list" parameter of the "set multiplex" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

The `InvalidParameter` exception is raised when the value of one of the parameters prohibits the normal execution of the action. The `completion_status` member indicates whether the action was completed (with a default value assigned to the inadequate parameter) or not. The `parameter_number` member identifies the rank of the invalid parameter.

7.1.12.2 `setDemultiplex` operation

Synopsis:

Interface:	`MhMultiplexedContent`	
Operation:	`setDemultiplex`	
Result:	`void`	
In:	`sequence<StreamIdentifier>`	`stream_list`
Exception:	`InvalidTarget`	
Exception:	`InvalidParameter`	

Description:

This operation specifies the demultiplexing of a multiple stream data of a multiplexed content object, e.g. a Moving Picture Experts Group (MPEG) stream, the result is set in a list of content objects which are generated if they do not exist yet. Each content object contains one demultiplexed stream.

The `setDemultiplex` operation triggers the execution of the "set demultiplex" elementary action with the bound multiplexed content object as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 41.2.2.

The `stream_list` parameter specifies the value of the "stream list" parameter of the "set demultiplex" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

The `InvalidParameter` exception is raised when the value of one of the parameters prohibits the normal execution of the action. The `completion_status` member indicates whether the action was completed (with a default value assigned to the inadequate parameter) or not. The `parameter_number` member identifies the rank of the invalid parameter.

7.1.12.3 IDL description

```
interface   MhMultiplexedContent: MhGenericContent  {

    void
        setMultiplex(
            in sequence<StreamIdentifier>
                stream_list)
    raises(InvalidTarget, InvalidParameter);

    void
        setDemultiplex(
            in sequence<StreamIdentifier>
                stream_list)
    raises(InvalidTarget, InvalidParameter);

};
```

7.1.13 `MhComposite` object

For the `MhComposite` object no specific operations are defined. The object inherits from the `MhComponent` object.

7.1.13.1 IDL description

```
interface   MhComposite: MhComponent {};
```

7.1.14 `MhScript` object

For the `MhScript` object no specific operations are defined. The object inherits from the `MhModel` object.

7.1.14.1 IDL description

```
interface   MhScript: MhModel {};
```

7.1.15 `MhContainer` object

For the `MhContainer` object no specific operations are defined. The object inherits from the `MhObject` object.

7.1.15.1 IDL description

```
interface   MhContainer: MhObject {};
```

7.1.16 `MhDescriptor` object

For the `MhDescriptor` object no specific operations are defined. The object inherits from the `MhObject` object.

7.1.16.1 IDL description

```
interface   MhDescriptor: MhObject {};
```

7.1.17 `RtObjectOrSocket` object

The following subclause defines the operations of the `RtObjectOrSocket` object.

7.1.17.1 `setGlobalBehaviour` operation

Synopsis:

Interface:	`RtObjectOrSocket`	
Operation:	`setGlobalBehaviour`	
Result:	`void`	
In:	`GlobalBehaviour`	`global_behaviour`
Exception:	`InvalidTarget`	
Exception:	`InvalidParameter`	

Description:

This operation enables the modification of the global behaviour of an rt-object or a socket.

The `setGlobalBehaviour` operation triggers the execution of the "set global behaviour" elementary action with the bound rt-object or socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 44.2.1.

The `global_behaviour` parameter specifies the value of the "global behaviour" parameter of the "set global behaviour" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

The `InvalidParameter` exception is raised when the value of one of the parameters prohibits the normal execution of the action. The `completion_status` member indicates whether the action was completed (with a default value assigned to the inadequate parameter) or not. The `parameter_number` member identifies the rank of the invalid parameter.

7.1.17.2 `getGlobalBehaviour` operation

Synopsis:

Interface:	`RtObjectOrSocket`
Operation:	`getGlobalBehaviour`
Result:	`GenericValue`
Exception:	`InvalidTarget`

Description:

This operation retrieves all the attributes value composing the global behaviour of each rt-object or socket to the MHEG engine.

The `getGlobalBehaviour` operation triggers the execution of the "get global behaviour" elementary action with the bound rt-object or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 44.3.1.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.17.3 `run` operation

Synopsis:

Interface:	`RtObjectOrSocket`
Operation:	`run`
Result:	`void`
Exception:	`InvalidTarget`

Description:

This operation enables the activation of the rt-object or the socket by the running process.

The `run` operation triggers the execution of the "run" elementary action with the bound rt-object or socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 45.2.1.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.17.4 `stop` operation

Synopsis:

Interface:	`RtObjectOrSocket`
Operation:	`stop`
Result:	`void`
Exception:	`InvalidTarget`

Description:

This operation removes the rt-object from the running process.

The `stop` operation triggers the execution of the "stop" elementary action with the bound rt-object as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 45.2.2.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.17.5 IDL description

```
interface   RtObjectOrSocket    {

    void
        setGlobalBehaviour(
            in GlobalBehaviour
                global_behaviour)
    raises(InvalidTarget, InvalidParameter);

    GenericValue
        getGlobalBehaviour()
    raises(InvalidTarget);

    void
        run()
    raises(InvalidTarget);

    void
        stop()
    raises(InvalidTarget);

};
```

7.1.18 `RtObject` object

The following subclause defines the operations of the `RtObject` object. The object inherits from the `RtObjectOrSocket` and from the `Entity` object.

7.1.18.1 `bind` operation

Synopsis:

Interface:	`RtObject`	
Operation:	`bind`	
Result:	`RtObjectIdentifier`	
In:	`RtObjectReference`	`rt_object_reference`
Exception:	`AlreadyBound`	
Exception:	`InvalidTarget`	

Description:

This operation binds the RtObject instance (an interface object instance) with an rt-object (an MHEG entity).

The `rt_object_reference` parameter specifies the reference of the rt-object.

The operation returns the identifier of the bound rt-object.

The `AlreadyBound` exception is raised when the interface object instance is already bound with an MHEG entity.

The `InvalidTarget` exception is raised when the targeted MHEG entity is not available. The `period` member returns the current period of the target.

7.1.18.2 `unbind` operation

Synopsis:

Interface:	`RtObject`
Operation:	`unbind`
Result:	`void`
Exception:	`NotBound`

Description:

This operation cancels the binding between the RtObject instance (an interface object instance) and an rt-object (an MHEG entity).

The `NotBound` exception is raised when the interface object instance is not bound with an MHEG entity.

7.1.18.3 `new` operation

Synopsis:

Interface:	`RtObject`	
Operation:	`new`	
Result:	`RtObjectIdentifier`	
In:	`RtObjectReference`	`rt_object_reference`
Exception:	`AlreadyBound`	
Exception:	`InvalidTarget`	

Description:

This operation enables the creation of an rt-object from a model object by the MHEG engine.

The `new` operation triggers the execution of the "new" elementary action targeted at a single rt-object.

The effect of the action on its target and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 43.2.1.

The `rt_object_reference` parameter specifies a reference to an rt-object.

This operation implicitly binds the RtObject instance (an interface object instance) with the new created rt-object (an MHEG entity).

The operation returns the identifier of the new created rt-object bound with the Rtobject instance.

The `AlreadyBound` exception is raised when the interface object instance is already bound with an MHEG entity.

The `InvalidTarget` exception is raised when the targeted MHEG entity is not available. The `period` member returns the current period of the target.

7.1.18.4 `delete` operation

Synopsis:

Interface:	`RtObject`
Operation:	`delete`
Result:	`void`
Exception:	`NotBound`
Exception:	`InvalidTarget`

Description:

This operation enables the removing of an rt-object by the MHEG engine.

The `delete` operation triggers the execution of the "delete" elementary action targeted at a single rt-object.

The effect of the action on its target and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 43.2.2.

This operation implicitly cancels the binding between the RtObject instance (an interface object instance) and the new deleted rt-object (an MHEG entity).

The `NotBound` exception is raised when the interface object instance is not bound with an MHEG entity.

The `InvalidTarget` exception is raised when the targeted MHEG entity is not available. The `period` member returns the current period of the target.

7.1.18.5 `getAvailabilityStatus` operation

Synopsis:

Interface:	`RtObject`
Operation:	`getAvailabilityStatus`
Result:	`RtAvailabilityStatusValue`
Exception:	`NotBound`
Exception:	`InvalidTarget`

Description:

This operation retrieves the availability of an rt-object to the MHEG engine.

The `getAvailabilityStatus` operation triggers the execution of the "get rt-availability status" elementary action with the bound rt-object as its single target.

The effect of the action on its target, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 43.3.1.

The operation returns the availability of the rt-object bound with the RtObject instance. The returned value is either `NOT_AVAILABLE`, `PROCESSING` or `AVAILABLE`.

When the returned value is `NOT_AVAILABLE`, the operation implicitly cancels the binding between the RtObject instance (an interface object instance) and the rt-object (an MHEG entity).

The `NotBound` exception is raised when the interface object instance is not bound with an MHEG entity.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.18.6 `getIdentifier` operation

Synopsis:

Interface:	`RtObject`
Operation:	`getIdentifier`
Result:	`RtObjectIdentifier`
Exception:	`NotBound`

Description:

This operation retrieves the identifier of the rt-object (an MHEG entity) bound with the RtObject instance (an interface object instance).

The `NotBound` exception is raised when the interface object instance is not bound with an MHEG entity.

7.1.18.7 `kill` operation

Synopsis:

Interface: `RtObject`
Operation: `kill`
Result: `void`

Description:

This operation deletes the RtObject instance (an interface object instance).

7.1.18.8 `getRunningStatus` operation

Synopsis:

Interface: `RtObject`
Operation: `getRunningStatus`
Result: `RunningStatusValue`
Exception: `InvalidTarget`

Description:

This operation get the activation of each rt-object and each socket by the MHEG engine.

The `getRunningStatus` operation triggers the execution of the "get running status" elementary action with the bound rt-object or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 45.3.1.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.18.9 IDL description

```
interface   RtObject: Entity, RtObjectOrSocket  {

    RtObjectIdentifier
        bind(
            in RtObjectReference
                rt_object_reference)
    raises(AlreadyBound, InvalidTarget);

    void
        unbind()
    raises(NotBound);

    RtObjectIdentifier
        new(
            in RtObjectReference
                rt_object_reference)
    raises(AlreadyBound, InvalidTarget);

    void
        delete()
    raises(NotBound, InvalidTarget);

    RtAvailabilityStatusValue
        getAvailabilityStatus()
    raises(NotBound, InvalidTarget);
```

```
    RtObjectIdentifier
        getIdentifier()
    raises(NotBound);

    void
        kill();


    RunningStatusValue
        getRunningStatus()
    raises(InvalidTarget);

};
```

7.1.19 `Socket` object

The following subclause defines the operations of the `Socket` object. The object inherits from the `RtObjectOrSocket` object, the RtComponentOrSocket object and from the `Entity` object.

7.1.19.1 `bind` operation

Synopsis:

Interface:	`Socket`	
Operation:	`bind`	
Result:	`SocketIdentification`	
In:	`SocketReference`	`socket_reference`
Exception:	`AlreadyBound`	
Exception:	`InvalidTarget`	

Description:

This operation binds the Socket instance (an interface object instance) with a socket (an MHEG entity).

The `socket_reference` parameter specifies the reference of the socket.

The operation returns the identifier of the bound socket.

The `AlreadyBound` exception is raised when the interface object instance is already bound with an MHEG entity.

The `InvalidTarget` exception is raised when the targeted MHEG entity is not available. The `period` member returns the current period of the target.

7.1.19.2 `unbind` operation

Synopsis:

Interface:	`Socket`
Operation:	`unbind`
Result:	`void`
Exception:	`NotBound`

Description:

This operation cancels the binding between the Socket instance (an interface object instance) and a socket (an MHEG entity).

The `NotBound` exception is raised when the interface object instance is not bound with an MHEG entity.

7.1.19.3 `getIdentifier` operation

Synopsis:

Interface:	`Socket`
Operation:	`getIdentifier`
Result:	`SocketIdentification`

Description:

This operation retrieves the identifier of the socket (an MHEG entity) bound with the Socket instance (an interface object instance).

The `NotBound` exception is raised when the interface object instance is not bound with an MHEG entity.

7.1.19.4 `kill` operation

Synopsis:

Interface:	`Socket`
Operation:	`kill`
Result:	`void`

Description:

This operation deletes the Socket instance (an interface object instance).

7.1.19.5 `plug` operation

Synopsis:

Interface:	`Socket`	
Operation:	`plug`	
Result:	`void`	
In:	`PlugIn`	`plug_in`
Exception:	`InvalidTarget`	

Description:

This operation enables a dynamism in the presentation and structure. This operation specifies the information to be plugged into a socket. This is used to obtain a different presentation or structure from the same composite object model.

The `plug` operation triggers the execution of the "plug" elementary action with the bound socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 48.2.1.

The `plug_in` parameter specifies the value of the "plug in" parameter of the "plug" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.19.6 `setVisibleDurationPosition` operation

Synopsis:

Interface:	`Socket`	
Operation:	`setVisibleDurationPosition`	
Result:	`void`	
In:	`VisibleDurationPosition`	`visible_duration_position`
Exception:	`InvalidTarget`	
Exception:	`InvalidParameter`	

Description:

This operation specifies within the perceptible duration of the parent the position where to attach the visible duration of a socket.

The `setVisibleDurationPosition` operation triggers the execution of the "set visible duration position" elementary action with the bound socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 52.2.4.

The `visible_duration_position` parameter specifies the value of the "parent relative generic space temporal position" parameter of the "set visible duration position" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

The `InvalidParameter` exception is raised when the value of one of the parameters prohibits the normal execution of the action. The `completion_status` member indicates whether the action was completed (with a default value assigned to the inadequate parameter) or not. The `parameter_number` member identifies the rank of the invalid parameter.

7.1.19.7 `getVisibleDurationPosition` operation

Synopsis:

Interface:	`Socket`
Operation:	`getVisibleDurationPosition`
Result:	`unsigned long`
Exception:	`InvalidTarget`

Description:

This operation retrieves the visible duration position value of the socket within its parent relative generic space. This value is retrieved in relative generic temporal unit.

The `getVisibleDurationPosition` operation triggers the execution of the "get VD position" elementary action with the bound socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 52.3.7.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.19.8 IDL description

```
interface   Socket: Entity, RtObjectOrSocket, RtComponentOrSocket   {

    SocketIdentification
        bind(
            in SocketReference
                socket_reference)
    raises(AlreadyBound, InvalidTarget);

    void    unbind()
        raises(NotBound);

    SocketIdentification
        getIdentifier();
    void
        kill();

    void
        plug(
            in PlugIn
                plug_in)
    raises(InvalidTarget);

    void
        setVisibleDurationPosition(
            in VisibleDurationPosition
                visible_duration_position)
    raises(InvalidTarget, InvalidParameter);

    unsigned long
        getVisibleDurationPosition()
    raises(InvalidTarget);

};
```

7.1.20 `RtScript` object

The following subclause defines the operations of the `RtScript` object. The object inherits from the `RtObject` object.

7.1.20.1 `setParameters` operation

Synopsis:

Interface:	`RtScript`	
Operation:	`setParameters`	
Result:	`void`	
In:	`sequence<Parameter>`	`parameters`
Exception:	`InvalidTarget`	

Description:

This operation enables parameter passing between rt-scripts and other MHEG entities.

The `setParameters` operation triggers the execution of the "set parameters" elementary action with the bound rt-script as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 46.2.1.

The `parameters` parameter specifies the value of the "parameters" parameter of the "set parameters" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.20.2 `getTerminationStatus` operation

Synopsis:

Interface: `RtScript`
Operation: `getTerminationStatus`
Result: `TerminationStatusValue`
Exception: `InvalidTarget`

Description:

This operation gets the process termination of each rt-script and by the script process.

The `getTerminationStatus` operation triggers the execution of the "get termination status" elementary action with the bound rt-script as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 47.3.1.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.20.3 IDL description

```
interface   RtScript: RtObject  {

    void
        setParameters(
            in sequence<Parameter>
                parameters)
    raises(InvalidTarget);

    TerminationStatusValue
        getTerminationStatus()
    raises(InvalidTarget);

};
```

7.1.21 `RtComponentOrSocket` object

The following subclause defines the operations of the `RtComponentOrSocket` object.

7.1.21.1 `setRGS` operation

Synopsis:

Interface: `RtComponentOrSocket`
Operation: `setRGS`
Result: `void`
In: `ChannelIdentifier` `channel_identifier`
Exception: `InvalidTarget`

Description:

This operation assigns an rt-component or a socket to a Relative Generic Space (RGS).

The `setRGS` operation triggers the execution of the "set RGS" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 50.2.1.

The `channel_identifier` parameter specifies the value of the "channel identifier" parameter of the "set RGS" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.2 `getRGS` operation

Synopsis:

Interface:	`RtComponentOrSocket`	
Operation:	`getRGS`	
Result:	`RGSValue`	
Exception:	`InvalidTarget`	

Description:

This operation retrieves the RGS assigned to an rt-component or to a socket.

The `getRGS` operation triggers the execution of the "get RGS" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 50.3.1.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.3 `setOpacity` operation

Synopsis:

Interface:	`RtComponentOrSocket`	
Operation:	`setOpacity`	
Result:	`void`	
In:	`unsigned short`	`opacity_rate`
In:	`unsigned long`	`transition_duration`
Exception:	`InvalidTarget`	

Description:

This operation assigns an opacity rate value to an rt-component or a socket.

The `setOpacity` operation triggers the execution of the "set opacity" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 51.2.1.

The `opacity_rate` parameter specifies the value of the "opacity rate" parameter of the "set opacity" action.

The `transition_duration` parameter specifies the value of the "transition duration" parameter of the "set opacity" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.4 `setPresentationPriority` operation

Synopsis:

Interface:	`RtComponentOrSocket`	
Operation:	`setPresentationPriority`	
Result:	`void`	
In:	`PresentationPriority`	`presentation_priority`
In:	`unsigned long`	`transition_duration`
Exception:	`InvalidTarget`	
Exception:	`InvalidParameter`	

Description:

This operation specifies the presentation priority between the rt-components or sockets assigned to the same RGS. The operation defines the priority of the rt-component or socket with respect to the other rt-components or sockets assigned to the same RGS.

The `setPresentationPriority` operation triggers the execution of the "set Presentation priority" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 51.2.2.

The `presentation_priority` parameter specifies the value of the "presentation priority" parameter of the "set Presentation priority" action.

The `transition_duration` parameter specifies the value of the "transition duration" parameter of the "set Presentation priority" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

The `InvalidParameter` exception is raised when the value of one of the parameters prohibits the normal execution of the action. The `completion_status` member indicates whether the action was completed (with a default value assigned to the inadequate parameter) or not. The `parameter_number` member identifies the rank of the invalid parameter.

7.1.21.5 `getOpacity` operation

Synopsis:

Interface:	`RtComponentOrSocket`
Operation:	`getOpacity`
Result:	`unsigned short`
Exception:	`InvalidTarget`

Description:

This operation retrieves the opacity value of an rt-component or a socket.

The `getOpacity` operation triggers the execution of the "get opacity" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 51.3.1.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.6 `getEffectiveOpacity` operation

Synopsis:

Interface:	`RtComponentOrSocket`
Operation:	`getEffectiveOpacity`
Result:	`unsigned short`
Exception:	`InvalidTarget`

Description:

This operation retrieves the effective opacity value of an rt-component or a socket.

The `getEffectiveOpacity` operation triggers the execution of the "get effective opacity" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 51.3.2.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.7 `getPresentationPriority` operation

Synopsis:

Interface:	`RtComponentOrSocket`
Operation:	`getPresentationPriority`
Result:	`unsigned short`
Exception:	`InvalidTarget`

Description:

This operation retrieves the presentation priority of an rt-component or a socket.

The `getPresentationPriority` operation triggers the execution of the "get presentation priority" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 51.3.3.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.8 `setVisibleDuration` operation

Synopsis:

Interface:	`RtComponentOrSocket`	
Operation:	`setVisibleDuration`	
Result:	`void`	
In:	`TemporalPosition`	`initial_temporal_position`
In:	`TemporalPosition`	`terminal_temporal_position`
Exception:	`InvalidTarget`	
Exception:	`InvalidParameter`	

Description:

This operation specifies the visible duration of an rt-component or a socket.

The `setVisibleDuration` operation triggers the execution of the "set visible duration" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 52.2.1.

The `initial_temporal_position` parameter specifies the value of the "initial temporal position" parameter of the "set visible duration" action.

The `terminal_temporal_position` parameter specifies the value of the "terminal temporal position" parameter of the "set visible duration" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

The `InvalidParameter` exception is raised when the value of one of the parameters prohibits the normal execution of the action. The `completion_status` member indicates whether the action was completed (with a default value assigned to the inadequate parameter) or not. The `parameter_number` member identifies the rank of the invalid parameter.

7.1.21.9 `setTemporalTermination` operation

Synopsis:

Interface:	`RtComponentOrSocket`	
Operation:	`setTemporalTermination`	
Result:	`void`	
In:	`TemporalTermination`	`temporal_termination`
Exception:	`InvalidTarget`	

Description:

This operation specifies the type of temporal termination when the current temporal position passes the terminal temporal position.

The `setTemporalTermination` operation triggers the execution of the "set temporal termination" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 52.2.2.

The `temporal_termination` parameter specifies the value of the "temporal termination" parameter of the "set temporal termination" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.10 `setCurrentTemporalPosition` operation

Synopsis:

Interface:	`RtComponentOrSocket`	
Operation:	`setCurrentTemporalPosition`	
Result:	`void`	
In:	`TemporalPosition`	`temporal_position`
Exception:	`InvalidTarget`	
Exception:	`InvalidParameter`	

Description:

This operation specifies a current temporal position within the Visible Duration (VD).

The `setCurrentTemporalPosition` operation triggers the execution of the "set current temporal position" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 52.2.3.

The `temporal_position` parameter specifies the value of the "temporal position" parameter of the "set current temporal position" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

The `InvalidParameter` exception is raised when the value of one of the parameters prohibits the normal execution of the action. The `completion_status` member indicates whether the action was completed (with a default value assigned to the inadequate parameter) or not. The `parameter_number` member identifies the rank of the invalid parameter.

7.1.21.11 `setSpeed` operation

Synopsis:

Interface:	`RtComponentOrSocket`	
Operation:	`setSpeed`	
Result:	`void`	
In:	`Speed`	`the_speed`
In:	`unsigned long`	`transition_duration`
Exception:	`InvalidTarget`	
Exception:	`InvalidParameter`	

Description:

This operation defines the speed of the presentation of an rt-component or socket. The effective presentation speed is calculated from this value by the MHEG engine.

The `setSpeed` operation triggers the execution of the "set speed" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 52.2.5.

The `the_speed` parameter specifies the value of the "speed" parameter of the "set speed" action.

The `transition_duration` parameter specifies the value of the "transition duration" parameter of the "set speed" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

The `InvalidParameter` exception is raised when the value of one of the parameters prohibits the normal execution of the action. The `completion_status` member indicates whether the action was completed (with a default value assigned to the inadequate parameter) or not. The `parameter_number` member identifies the rank of the invalid parameter.

7.1.21.12 `setTimestones` operation

Synopsis:

Interface:	`RtComponentOrSocket`	
Operation:	`setTimestones`	
Result:	`void`	
In:	`sequence<Timestone>`	`timestones`
Exception:	`InvalidTarget`	
Exception:	`InvalidParameter`	

Description:

This operation specifies a complete set of temporal markers within the perceptible duration of the presentable.

The `setTimestones` operation triggers the execution of the "set timestones" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 52.2.6.

The `timestones` parameter specifies the value of the "timestones" parameter of the "set timestones" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

The `InvalidParameter` exception is raised when the value of one of the parameters prohibits the normal execution of the action. The `completion_status` member indicates whether the action was completed (with a default value assigned to the inadequate parameter) or not. The `parameter_number` member identifies the rank of the invalid parameter.

7.1.21.13 `getInitialTemporalPosition` operation

Synopsis:

Interface:	`RtComponentOrSocket`
Operation:	`getInitialTemporalPosition`
Result:	`unsigned long`
Exception:	`InvalidTarget`

Description:

This operation retrieves the initial temporal position value of the rt-component or socket. This value is retrieved in Original generic space Generic Temporal Unit (OGTU).

The `getInitialTemporalPosition` operation triggers the execution of the "get initial temporal position" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 52.3.2.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.14 `getTerminalTemporalPosition` operation

Synopsis:

Interface:	`RtComponentOrSocket`
Operation:	`getTerminalTemporalPosition`
Result:	`unsigned long`
Exception:	`InvalidTarget`

Description:

This operation retrieves the terminal temporal position value of the rt-component or socket. This value is retrieved in OGTU.

The `getTerminalTemporalPosition` operation triggers the execution of the "get terminal temporal position" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 52.3.3.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.15 `getVDLength` operation

Synopsis:

Interface:	`RtComponentOrSocket`	
Operation:	`getVDLength`	
Result:	`unsigned long`	
In:	`GTIndicator`	`gt_indicator`
Exception:	`InvalidTarget`	

Description:

This operation retrieves the VD length value of the rt-component or socket either in OGTU or in Relative generic space Generic Temporal Unit (RGTU).

The `getVDLength` operation triggers the execution of the "get VD length" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 52.3.4.

The `gt_indicator` parameter specifies the value of the "GT indicator" parameter of the "get VD length" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.16 `getTemporalTermination` operation

Synopsis:

Interface:	`RtComponentOrSocket`
Operation:	`getTemporalTermination`
Result:	`TemporalTermination`
Exception:	`InvalidTarget`

Description:

This operation retrieves the "temporal termination" value of the rt-component or socket.

The `getTemporalTermination` operation triggers the execution of the "get temporal termination" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 52.3.5.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.17 `getCurrentTemporalPosition` operation

Synopsis:

Interface:	`RtComponentOrSocket`
Operation:	`getCurrentTemporalPosition`
Result:	`unsigned long`
Exception:	`InvalidTarget`

Description:

This operation retrieves the current temporal position value of the rt-component or socket. This value is retrieved in OGTU.

The `getCurrentTemporalPosition` operation triggers the execution of the "get current temporal position" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 52.3.6.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.18 `getSpeedRate` operation

Synopsis:

Interface:	`RtComponentOrSocket`
Operation:	`getSpeedRate`
Result:	`short`
Exception:	`InvalidTarget`

Description:

This operation retrieves the speed rate value of the rt-component or socket. This value is a percentage negative or positive. This speed rate, is used to indicate the required change of speed since the Initial Original generic space Generic Temporal Ratio (IOGTR) and also the required direction of the presentation.

The `getSpeedRate` operation triggers the execution of the "get speed rate" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 52.3.8.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.19 `getOGTR` operation

Synopsis:

Interface:	`RtComponentOrSocket`
Operation:	`getOGTR`
Result:	`unsigned long`
Exception:	`InvalidTarget`

Description:

This operation retrieves the Original generic space Generic Temporal Ratio (OGTR) value of the rt-component or socket. This value is a positive or null numeric which corresponds to the number of OGTU to be mapped in one second.

The `getOGTR` operation triggers the execution of the "get OGTR" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 52.3.9.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.20 `getEffectiveSpeedRate` operation

Synopsis:

Interface:	`RtComponentOrSocket`
Operation:	`getEffectiveSpeedRate`
Result:	`short`
Exception:	`InvalidTarget`

Description:

This operation retrieves the effective speed rate value of the rt-component or socket. This value is a percentage negative or positive. This effective speed rate, is used to calculate the effective change of speed since the IOGTR and also the effective direction of the presentation.

The `getEffectiveSpeedRate` operation triggers the execution of the "get effective speed rate" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 52.3.10.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.21 `getEffectiveOGTR` operation

Synopsis:

Interface:	`RtComponentOrSocket`
Operation:	`getEffectiveOGTR`
Result:	`unsigned long`
Exception:	`InvalidTarget`

Description:

This operation retrieves the effective OGTR value of the rt-component or socket. This value is a positive or null numeric which corresponds to the effective number of OGTU to be mapped in one second.

The `getEffectiveOGTR` operation triggers the execution of the "get effective OGTR elementary action" with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 52.3.11.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.22 `getTimestoneStatus` operation

Synopsis:

Interface:	`RtComponentOrSocket`	
Operation:	`getTimestoneStatus`	
Result:	`unsigned short`	
Exception:	`InvalidTarget`	

Description:

This operation retrieves the timestone status value of the rt-component or socket.

The `getTimestoneStatus` operation triggers the execution of the "get timestone status" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 52.3.12.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.23 `setPerceptibleSizeProjection` operation

Synopsis:

Interface:	`RtComponentOrSocket`	
Operation:	`setPerceptibleSizeProjection`	
Result:	`void`	
In:	`PerceptibleSizeProjection`	`perceptible_size_projection`
In:	`unsigned long`	`transition_duration`
Exception:	`InvalidTarget`	
Exception:	`InvalidParameter`	

Description:

This operation defines the projection of the perceptible size in its RGS.

The `setPerceptibleSizeProjection` operation triggers the execution of the "set perceptible size projection" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 53.2.1.

The `perceptible_size_projection` parameter specifies the value of the "perceptible size projection" parameter of the "set perceptible size projection" action.

The `transition_duration` parameter specifies the value of the "transition duration" parameter of the "set perceptible size projection" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

The `InvalidParameter` exception is raised when the value of one of the parameters prohibits the normal execution of the action. The `completion_status` member indicates whether the action was completed (with a default value assigned to the inadequate parameter) or not. The `parameter_number` member identifies the rank of the invalid parameter.

7.1.21.24 `setAspectRatio` operation

Synopsis:

Interface:	`RtComponentOrSocket`	
Operation:	`setAspectRatio`	
Result:	`void`	
In:	`AspectRatio`	`preserved`
Exception:	`InvalidTarget`	

Description:

This operation specifies whether in performing the projection of an rt-component or socket in the Channel Generic Space (CGS) (through the chain of mappings Original Generic Space (OGS)-RGS), the ratio between the Perceptible Size (PS) in Original generic space Generic Spatial Unit (OGSU), i.e. the OGS lengths, and the projection in Channel Generic Space Unit (CGSU) of the "size of the content information" is to be the same for each axis. In such case the aspect ratio is preserved.

The `setAspectRatio` operation triggers the execution of the "set aspect ratio preserved" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 53.2.3.

The `preserved` parameter specifies the value of the "preserved" parameter of the "set aspect ratio preserved" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.25 `setVisibleSize` operation

Synopsis:

Interface:	`RtComponentOrSocket`	
Operation:	`setVisibleSize`	
Result:	`void`	
In:	`VSGS`	`the_vsgs`
In:	`VS`	`the_vs`
In:	`unsigned long`	`transition_duration`
Exception:	`InvalidTarget`	
Exception:	`InvalidParameter`	

Description:

This operation specifies the Visible Size (VS), which defines which portion of the PS is perceived by the user.

The `setVisibleSize` operation triggers the execution of the "set visible size" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 53.2.4.

The `the_vsgs` parameter specifies the value of the "vsgs" parameter of the "set visible size" action.

The `the_vs` parameter specifies the value of the "vs" parameter of the "set visible size" action.

The `transition_duration` parameter specifies the value of the "transition duration" parameter of the "set visible size" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

The `InvalidParameter` exception is raised when the value of one of the parameters prohibits the normal execution of the action. The `completion_status` member indicates whether the action was completed (with a default value assigned to the inadequate parameter) or not. The `parameter_number` member identifies the rank of the invalid parameter.

7.1.21.26 `setVisibleSizesAdjustment` operation

Synopsis:

Interface:	`RtComponentOrSocket`	
Operation:	`setVisibleSizesAdjustment`	
Result:	`void`	
In:	`sequence<AdjustmentAxis>`	`set_of_axes`
In:	`AdjustmentPolicy`	`adjustment_policy`
In:	`unsigned long`	`transition_duration`
Exception:	`InvalidTarget`	

Description:

This operation specifies the adjustment of a set of VSs on a same axis or on a set of axes. All the VSs to be adjusted need to be assigned to the same CGS.

The `setVisibleSizesAdjustment` operation triggers the execution of the "set visible sizes adjustment" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 53.2.5.

The `set_of_axes` parameter specifies the value of the "set of axes" parameter of the "set visible sizes adjustment" action.

The `adjustment_policy` parameter specifies the value of the "adjustment policy" parameter of the "set visible sizes adjustment" action.

The `transition_duration` parameter specifies the value of the "transition duration" parameter of the "set visible sizes adjustment" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.27 `setBox` operation

Synopsis:

Interface:	`RtComponentOrSocket`	
Operation:	`setBox`	
Result:	`void`	
In:	`BoxConstants`	`box`
Exception:	`InvalidTarget`	

Description:

This operation specifies whether the rt-component or the socket is presented with a box to show the perimeter of the VS.

The `setBox` operation triggers the execution of the "set box" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 53.2.6.

The `box` parameter specifies the value of the "box" parameter of the "set box" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.28 `setDefaultBackground` operation

Synopsis:

Interface:	`RtComponentOrSocket`	
Operation:	`setDefaultBackground`	
Result:	`void`	
In:	`unsigned short`	`background`
In:	`unsigned long`	`transition_duration`
Exception:	`InvalidTarget`	
Exception:	`InvalidParameter`	

Description:

This operation specifies whether the areas within the VS of an rt-component or a socket which are not filled by the presentation process need to be considered as opaque or transparent.

The `setDefaultBackground` operation triggers the execution of the "set default background" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 53.2.7.

The `background` parameter specifies the value of the "background" parameter of the "set default background" action.

The `transition_duration` parameter specifies the value of the "transition duration" parameter of the "set default background" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

The `InvalidParameter` exception is raised when the value of one of the parameters prohibits the normal execution of the action. The `completion_status` member indicates whether the action was completed (with a default value assigned to the inadequate parameter) or not. The `parameter_number` member identifies the rank of the invalid parameter.

7.1.21.29 `setAttachmentPoint` operation

Synopsis:

Interface:	`RtComponentOrSocket`	
Operation:	`setAttachmentPoint`	
Result:	`void`	
In:	`AttachmentPointType`	`type`
In:	`AttachmentPoint`	`positions`
Exception:	`InvalidTarget`	
Exception:	`InvalidParameter`	

Description:

This operation specifies one of the following AP: Perceptible Size Attachment Point (PSAP), Visible Size Internal Attachment Point (VSIAP) or Visible Size External Attachment Point (VSEAP).

The `setAttachmentPoint` operation triggers the execution of the "set attachment point" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 53.2.8.

The `type` parameter specifies the value of the "type" parameter of the "set attachment point" action.

The `positions` parameter specifies the value of the "positions" parameter of the "set attachment point" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

The `InvalidParameter` exception is raised when the value of one of the parameters prohibits the normal execution of the action. The `completion_status` member indicates whether the action was completed (with a default value assigned to the inadequate parameter) or not. The `parameter_number` member identifies the rank of the invalid parameter.

7.1.21.30 `setAttachmentPointPosition` operation

Synopsis:

Interface:	`RtComponentOrSocket`	
Operation:	`setAttachmentPointPosition`	
Result:	`void`	
In:	`AttachmentPointType`	`type`
In:	`ReferenceType`	`vseap_reference_point`
In:	`Lengths`	`the_lengths`
In:	`unsigned long`	`transition_duration`
Exception:	`InvalidTarget`	
Exception:	`InvalidParameter`	

Description:

This operation specifies the position of the VSIAP relatively to the PSAP, or the position of the VSEAP in its RGS relatively to the origin or to another VSEAP.

The `setAttachmentPointPosition` operation triggers the execution of the "set attachment point position" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 53.2.9.

The `type` parameter specifies the value of the "type" parameter of the "set attachment point position" action.

The `vseap_reference_point` parameter specifies the value of the "vseap reference point" parameter of the "set attachment point position" action.

The `the_lengths` parameter specifies the value of the "lengths" parameter of the "set attachment point position" action.

The `transition_duration` parameter specifies the value of the "transition duration" parameter of the "set attachment point position" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

The `InvalidParameter` exception is raised when the value of one of the parameters prohibits the normal execution of the action. The `completion_status` member indicates whether the action was completed (with a default value assigned to the inadequate parameter) or not. The `parameter_number` member identifies the rank of the invalid parameter.

7.1.21.31 `setVisibleSizesAlignment` operation

Synopsis:

Interface:	`RtComponentOrSocket`	
Operation:	`setVisibleSizesAlignment`	
Result:	`void`	
In:	`SizeBorder`	`size_border`
In:	`long`	`interval`
In:	`unsigned long`	`transition_duration`
Exception:	`InvalidTarget`	

Description:

This operation specifies the alignment of a set of VSs. The VSs are aligned on a border and an interval between two VSs may be provided. All the VSs to be aligned needs to be assigned to the same CGS.

The `setVisibleSizesAlignment` operation triggers the execution of the "set visible sizes alignment" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 53.2.10.

The `size_border` parameter specifies the value of the "size border" parameter of the "set visible sizes alignment" action.

The `interval` parameter specifies the value of the "interval" parameter of the "set visible sizes alignment" action.

The `transition_duration` parameter specifies the value of the "transition duration" parameter of the "set visible sizes alignment" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.32 `setMovingAbility` operation

Synopsis:

Interface:	`RtComponentOrSocket`	
Operation:	`setMovingAbility`	
Result:	`void`	
In:	`UserControls`	`moving_ability`
Exception:	`InvalidTarget`	

Description:

This operation specifies whether the user is able to move or not to move the VS of the targets in their CGS.

The `setMovingAbility` operation triggers the execution of the "set moving ability" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 53.2.11.

The `moving_ability` parameter specifies the value of the "moving ability" parameter of the "set moving ability" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.33 `setResizingAbility` operation

Synopsis:

Interface:	`RtComponentOrSocket`	
Operation:	`setResizingAbility`	
Result:	`void`	
In:	`UserControls`	`resizing_ability`
Exception:	`InvalidTarget`	

Description:

This operation specifies whether the user is able to resize or not the VS of the targets in their CGS.

The `setResizingAbility` operation triggers the execution of the "set resizing ability" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 53.2.12.

The `resizing_ability` parameter specifies the value of the "resizing ability" parameter of the "set resizing ability" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.34 `setScalingAbility` operation

Synopsis:

Interface:	`RtComponentOrSocket`	
Operation:	`setScalingAbility`	
Result:	`void`	
In:	`UserControls`	`scaling_ability`
Exception:	`InvalidTarget`	

Description:

This operation specifies whether the user is able to scale or not the PS of the targets in their CGS.

The `setScalingAbility` operation triggers the execution of the "set scaling ability" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 53.2.13.

The `scaling_ability` parameter specifies the value of the "scaling ability" parameter of the "set scaling ability" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.35 `setScrollingAbility` operation

Synopsis:

Interface:	`RtComponentOrSocket`	
Operation:	`setScrollingAbility`	
Result:	`void`	
In:	`UserControls`	`scrolling_ability`
Exception:	`InvalidTarget`	

Description:

This operation specifies whether the user is able/unable to scroll the PS through the VS of the targets in their CGS.

The `setScrollingAbility` operation triggers the execution of the "set scrolling ability" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 53.2.14.

The `scrolling_ability` parameter specifies the value of the "scrolling ability" parameter of the "set scrolling ability" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.36 `getGSR` operation

Synopsis:

Interface:	`RtComponentOrSocket`
Operation:	`getGSR`
Result:	`unsigned short`
Exception:	`InvalidTarget`

Description:

This operation retrieves the Generic Spatial Ratio (GSR) value of the OGS of the rt-component or socket. This ratio defines the number of OGSU which are to be mapped in one Relative generic space Generic Spatial Unit (RGSU).

The `getGSR` operation triggers the execution of the "get GSR" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 53.4.1.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.37 `getPS` operation

Synopsis:

Interface:	`RtComponentOrSocket`	
Operation:	`getPS`	
Result:	`SpecifiedPosition`	
In:	`GSIndicator`	`gs`
Exception:	`InvalidTarget`	

Description:

This operation retrieves the PS value of the rt-component or socket either in OGSU or in RGSU.

The `getPS` operation triggers the execution of the "get PS" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 53.4.2.

The `gs` parameter specifies the value of the "gs" parameter of the "get PS" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.38 `getAspectRatio` operation

Synopsis:

Interface:	`RtComponentOrSocket`
Operation:	`getAspectRatio`
Result:	`AspectRatio`
Exception:	`InvalidTarget`

Description:

This operation retrieves the aspect ratio value of the PS of the rt-component or socket.

The `getAspectRatio` operation triggers the execution of the "get aspect ratio" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 53.4.4.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.39 `getPSAP` operation

Synopsis:

Interface:	`RtComponentOrSocket`	
Operation:	`getPSAP`	
Result:	`SpecifiedPosition`	
In:	`PointType`	`point_type`
Exception:	`InvalidTarget`	

Description:

This operation retrieves the PSAP value of the rt-component or socket.

The `getPSAP` operation triggers the execution of the "get PSAP" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 53.4.5.

The `point_type` parameter specifies the value of the "point type" parameter of the "get PSAP" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.40 `getVSGS` operation

Synopsis:

Interface:	`RtComponentOrSocket`
Operation:	`getVSGS`
Result:	`VSGS`
Exception:	`InvalidTarget`

Description:

This operation retrieves the Visible Size Generic Space (VSGS) value of the rt-component or socket.

The `getVSGS` operation triggers the execution of the "get VSGS" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 53.4.6.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.41 `getVS` operation

Synopsis:

Interface:	`RtComponentOrSocket`
Operation:	`getVS`
Result:	`SpecifiedPosition`
Exception:	`InvalidTarget`

Description:

This operation retrieves the VS value of the rt-component or socket in Visible Size Generic Spatial Unit (VSGSU).

The `getVS` operation triggers the execution of the "get v s" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 53.4.7.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.42 `getBox` operation

Synopsis:

Interface: `RtComponentOrSocket`
Operation: `getBox`
Result: `BoxConstants`
Exception: `InvalidTarget`

Description:

This operation retrieves the visible size box value of the rt-component or socket.

The `getBox` operation triggers the execution of the "get box" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 53.4.8.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.43 `getDefaultBackground` operation

Synopsis:

Interface: `RtComponentOrSocket`
Operation: `getDefaultBackground`
Result: `unsigned short`
Exception: `InvalidTarget`

Description:

This operation retrieves the default background value of the VS of the rt-component or socket.

The `getDefaultBackground` operation triggers the execution of the "get default background" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 53.4.9.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.44 `getVSIAP` operation

Synopsis:

Interface:	`RtComponentOrSocket`	
Operation:	`getVSIAP`	
Result:	`SpecifiedPosition`	
In:	`PointType`	`point_type`
Exception:	`InvalidTarget`	

Description:

This operation retrieves the VSIAP value of the rt-component or socket.

The `getVSIAP` operation triggers the execution of the "get VSIAP" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 53.4.10.

The `point_type` parameter specifies the value of the "point type" parameter of the "get VSIAP" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.45 `getVSIAPPosition` operation

Synopsis:

Interface:	`RtComponentOrSocket`
Operation:	`getVSIAPPosition`
Result:	`SpecifiedPosition`
Exception:	`InvalidTarget`

Description:

This operation retrieves the VSIAP position value of the rt-component or socket. This is used to position the VSIAP relative to the PSAP.

The `getVSIAPPosition` operation triggers execution of the "get VSIAP position" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 53.4.11.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.46 `getVSEAP` operation

Synopsis:

Interface:	`RtComponentOrSocket`	
Operation:	`getVSEAP`	
Result:	`SpecifiedPosition`	
In:	`PointType`	`point_type`
Exception:	`InvalidTarget`	

Description:

This operation retrieves the VSEAP value of the rt-component or socket.

The `getVSEAP` operation triggers the execution of the "get VSEAP" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 53.4.12.

The `point_type` parameter specifies the value of the "point type" parameter of the "get VSEAP" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.47 `getVSEAPPosition` operation

Synopsis:

Interface:	`RtComponentOrSocket`	
Operation:	`getVSEAPPosition`	
Result:	`SpecifiedPosition`	
In:	`ReferencePoint`	`reference_point`
Exception:	`InvalidTarget`	

Description:

This operation retrieves the VSEAP position value of the rt-component or socket relative to a reference point. This is used to position the VSEAP relative to the PSAP.

The `getVSEAPPosition` operation triggers the execution of the "get VSEAP position" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 53.4.13.

The `reference_point` parameter specifies the value of the "reference point" parameter of the "get VSEAP position" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.48 `getMovingAbility` operation

Synopsis:

Interface:	`RtComponentOrSocket`
Operation:	`getMovingAbility`
Result:	`UserControls`
Exception:	`InvalidTarget`

Description:

This operation retrieves the moving ability value of the rt-component or socket.

The `getMovingAbility` operation triggers the execution of the "get moving ability" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 53.4.14.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.49 `getResizingAbility` operation

Synopsis:

Interface: `RtComponentOrSocket`
Operation: `getResizingAbility`
Result: `UserControls`
Exception: `InvalidTarget`

Description:

This operation retrieves the resizing ability value of the rt-component or socket.

The `getResizingAbility` operation triggers the execution of the "get resizing ability" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 53.4.15.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.50 `getScalingAbility` operation

Synopsis:

Interface: `RtComponentOrSocket`
Operation: `getScalingAbility`
Result: `UserControls`
Exception: `InvalidTarget`

Description:

This operation retrieves the scaling ability value of the rt-component or socket.

The `getScalingAbility` operation triggers the execution of the "get scaling ability" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 53.4.16.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.51 `getScrollingAbility` operation

Synopsis:

Interface: `RtComponentOrSocket`
Operation: `getScrollingAbility`
Result: `UserControls`
Exception: `InvalidTarget`

Description:

This operation retrieves the scrolling ability value of the rt-component or socket.

The `getScrollingAbility` operation triggers the execution of the "get scrolling ability" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 53.4.17.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.52 `setSelectability` operation

Synopsis:

Interface:	`RtComponentOrSocket`	
Operation:	`setSelectability`	
Result:	`void`	
In:	`unsigned short`	`min_number_of_selections`
In:	`unsigned short`	`max_number_of_selections`
Exception:	`InvalidTarget`	
Exception:	`InvalidParameter`	

Description:

This operation assigns a "minimum number of selections required" value and a "maximum number of selections required" value to an rt-component or a socket. The MHEG engine calculates the "selectability" value of the rt-component or socket from these two values. The "effective selectability" value of the rt-component or socket is also calculated by the MHEG engine from this "selectability" value and the "effective selectability" value of the rt-component or socket parent.

The `setSelectability` operation triggers the execution of the "set selectability" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 57.2.1.

The `min_number_of_selections` parameter specifies the value of the "min number of selections" parameter of the "set selectability" action.

The `max_number_of_selections` parameter specifies the value of the "max number of selections" parameter of the "set selectability" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

The `InvalidParameter` exception is raised when the value of one of the parameters prohibits the normal execution of the action. The `completion_status` member indicates whether the action was completed (with a default value assigned to the inadequate parameter) or not. The `parameter_number` member identifies the rank of the invalid parameter.

7.1.21.53 `setSelectionStatus` operation

Synopsis:

Interface:	`RtComponentOrSocket`	
Operation:	`setSelectionStatus`	
Result:	`void`	
In:	`SelectionStatusValue`	`selection_state`
Exception:	`InvalidTarget`	

Description:

This operation assigns a value to the "selection status" of an rt-component or a socket.

The `setSelectionStatus` operation triggers the execution of the "set selection status" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 57.2.2.

The `selection_state` parameter specifies the value of the "selection state" parameter of the "set selection status" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.54 `setSelectionPresentationEffectResponsibility` operation

Synopsis:

Interface:	`RtComponentOrSocket`	
Operation:	`setSelectionPresentationEffectResponsibility`	
Result:	`void`	
In:	`Responsibility`	`the_responsibility`
Exception:	`InvalidTarget`	

Description:

This operation assigns a value to the "selection presentation effect responsibility" of an rt-component or a socket. This attribute indicates if it is the MHEG engine or the author who is responsible for reflecting a new state of the rt-component or socket as its single target.

The `setSelectionPresentationEffectResponsibility` operation triggers the execution of the "set selection presentation effect responsibility" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 57.2.3.

The `the_responsibility` parameter specifies the value of the "responsibility" parameter of the "set selection presentation effect responsibility" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.55 `getSelectability` operation

Synopsis:

Interface:	`RtComponentOrSocket`	
Operation:	`getSelectability`	
Result:	`void`	
Out:	`unsigned short`	`min_mumber_of_selections`
Out:	`unsigned short`	`max_mumber_of_selections`
Exception:	`InvalidTarget`	

Description:

This operation retrieves the "minimum number of selections required" and the "maximum number of selections required". If the "maximum number of selections required" is equal to 0 the "selectability" of the rt-component or socket is "not selectable".

The `getSelectability` operation triggers the execution of the "get selectability" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 57.3.1.

The `min_mumber_of_selections` parameter specifies the value of the "min number of selections" parameter of the "get selectability" action.

The `max_mumber_of_selections` parameter specifies the value of the "max number of selections" parameter of the "get selectability" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.56 `getEffectiveSelectability` operation

Synopsis:

Interface:	`RtComponentOrSocket`
Operation:	`getEffectiveSelectability`
Result:	`EffectiveSelectability`
Exception:	`InvalidTarget`

Description:

This operation retrieves the "effective selectability" attribute value of the rt-component or socket.

The `getEffectiveSelectability` operation triggers the execution of the "get effective selectability" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 57.3.2.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.57 `getSelectionStatus` operation

Synopsis:

Interface: `RtComponentOrSocket`
Operation: `getSelectionStatus`
Result: `SelectionStatusValue`
Exception: `InvalidTarget`

Description:

This operation retrieves the "selection status" value of the rt-component or socket.

The `getSelectionStatus` operation triggers the execution of the "get selection status" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 57.3.3.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.58 `getSelectionMode` operation

Synopsis:

Interface: `RtComponentOrSocket`
Operation: `getSelectionMode`
Result: `SelectionModeValue`
Exception: `InvalidTarget`

Description:

This operation retrieves the "selection mode" attribute value of the rt-component or socket.

The `getSelectionMode` operation triggers the execution of the "get selection mode" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 57.3.4.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.59 `getSelectionPresentationEffectResponsibility` operation

Synopsis:

Interface: `RtComponentOrSocket`
Operation: `getSelectionPresentationEffectResponsibility`
Result: `Responsibility`
Exception: `InvalidTarget`

Description:

This operation retrieves the "selection presentation effect responsibility" attribute value of the rt-component or socket.

The `getSelectionPresentationEffectResponsibility` operation triggers the execution of the "get selection presentation effect responsibility" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 57.3.6.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.60 `setModifiability` operation

Synopsis:

Interface:	`RtComponentOrSocket`	
Operation:	`setModifiability`	
Result:	`void`	
In:	`unsigned short`	`min_number_of_modifications`
In:	`unsigned short`	`max_number_of_modifications`
Exception:	`InvalidTarget`	
Exception:	`InvalidParameter`	

Description:

This operation assigns a "minimum number of modifications required" value and a "maximum number of modifications required" value to an rt-component or a socket. The MHEG engine calculates the "modifiability" value of the rt-component or socket from these two values. The "effective modifiability" value of the rt-component or socket is also calculated by the MHEG engine from this "modifiability" value and the "effective modifiability" value of the parent of the rt-component or socket.

The `setModifiability` operation triggers the execution of the "set modifiability" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 58.2.1.

The `min_number_of_modifications` parameter specifies the value of the "min number of modifications" parameter of the "set modifiability" action.

The `max_number_of_modifications` parameter specifies the value of the "max number of modifications" parameter of the "set modifiability" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

The `InvalidParameter` exception is raised when the value of one of the parameters prohibits the normal execution of the action. The `completion_status` member indicates whether the action was completed (with a default value assigned to the inadequate parameter) or not. The `parameter_number` member identifies the rank of the invalid parameter.

7.1.21.61 `setModificationStatus` operation

Synopsis:

Interface:	`RtComponentOrSocket`	
Operation:	`setModificationStatus`	
Result:	`void`	
In:	`ModificationStatusValue`	`modification_state`
Exception:	`InvalidTarget`	

Description:

This operation assigns a value to the "modification status" of an rt-component or a socket.

The `setModificationStatus` operation triggers the execution of the "set modification status" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 58.2.2.

The `modification_state` parameter specifies the value of the "modification state" parameter of the "set modification status" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.62 `setModificationPresentationEffectResponsibility` operation

Synopsis:

Interface:	`RtComponentOrSocket`	
Operation:	`setModificationPresentationEffectResponsibility`	
Result:	`void`	
In:	`Responsibility`	`the_responsibility`
Exception:	`InvalidTarget`	

Description:

This operation assigns a value to the "modification presentation effect responsibility" of an rt-component or a socket. This attribute indicates if it is the MHEG engine or the author who is responsible for reflecting a new state of the component or socket.

The `setModificationPresentationEffectResponsibility` operation triggers the execution of the "set modification presentation effect responsibility" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 58.2.3.

The `the_responsibility` parameter specifies the value of the "responsibility" parameter of the "set modification presentation effect responsibility" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.63 `getModifiability` operation

Synopsis:

Interface:	`RtComponentOrSocket`	
Operation:	`getModifiability`	
Result:	`void`	
Out:	`unsigned short`	`min_numbers_of_modifications`
Out:	`unsigned short`	`max_numbers_of_modifications`
Exception:	`InvalidTarget`	

Description:

This operation retrieves the "minimum number of modifications required" and the "maximum number of modifications required". If the "maximum number of modifications required" is equal to 0 the "modifiability" of the rt-component or socket is "not modifiable".

The `getModifiability` operation triggers the execution of the "get modifiability" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 58.3.1.

The `min_numbers_of_modifications` parameter specifies the value of the "min numbers of modifications" parameter of the "get modifiability" action.

The `max_numbers_of_modifications` parameter specifies the value of the "max numbers of modifications" parameter of the "get modifiability" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.64 `getEffectiveModifiability` operation

Synopsis:

Interface:	`RtComponentOrSocket`
Operation:	`getEffectiveModifiability`
Result:	`EffectiveModifiability`
Exception:	`InvalidTarget`

Description:

This operation retrieves the "effective modifiability" attribute value of the rt-component or socket.

The `getEffectiveModifiability` operation triggers the execution of the "get effective modifiability" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 58.3.2.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.65 `getModificationStatus` operation

Synopsis:

Interface: `RtComponentOrSocket`
Operation: `getModificationStatus`
Result: `ModificationStatusValue`
Exception: `InvalidTarget`

Description:

This operation retrieves the "modification status" value of the rt-component or socket.

The `getModificationStatus` operation triggers the execution of the "get modification status" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 58.3.3.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.66 `getModificationMode` operation

Synopsis:

Interface: `RtComponentOrSocket`
Operation: `getModificationMode`
Result: `ModificationModeValue`
Exception: `InvalidTarget`

Description:

This operation retrieves the "modification mode" attribute value of the rt-component or socket.

The `getModificationMode` operation triggers the execution of the "get modification mode" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 58.3.4.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.67 `getModificationPresentationEffectResponsibility` operation

Synopsis:

Interface: `RtComponentOrSocket`
Operation: `getModificationPresentationEffectResponsibility`
Result: `Responsibility`
Exception: `InvalidTarget`

Description:

This operation retrieves the "modification presentation effect responsibility" attribute value of the rt-component or socket.

The `getModificationPresentationEffectResponsibility` operation triggers the execution of the "get modification presentation effect responsibility" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 58.3.6.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.68 `setNoInteractionStyle` operation

Synopsis:

Interface: `RtComponentOrSocket`
Operation: `setNoInteractionStyle`
Result: `void`
Exception: `InvalidTarget`

Description:

This operation deassigns the currently assigned interaction style to an rt-component or a socket.

The `setNoInteractionStyle` operation triggers the execution of the "set no style" elementary action with the bound rt-component or socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 59.2.6.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.21.69 IDL description

```
interface   RtComponentOrSocket {

    void
        setRGS(
            in ChannelIdentifier
                channel_identifier)
    raises(InvalidTarget);

    RGSValue
        getRGS()
    raises(InvalidTarget);

    void
        setOpacity(
            in unsigned short
                opacity_rate,
            in unsigned long
                transition_duration)
    raises(InvalidTarget);

    void
        setPresentationPriority(
            in PresentationPriority
                presentation_priority,
            in unsigned long
                transition_duration)
    raises(InvalidTarget, InvalidParameter);

    unsigned short
        getOpacity()
```

```
    raises(InvalidTarget);

    unsigned short
        getEffectiveOpacity()
    raises(InvalidTarget);

    unsigned short
        getPresentationPriority()
    raises(InvalidTarget);

    void
        setVisibleDuration(
            in TemporalPosition
                initial_temporal_position,
            in TemporalPosition
                terminal_temporal_position)
    raises(InvalidTarget, InvalidParameter);

    void
        setTemporalTermination(
            in TemporalTermination
                temporal_termination)
    raises(InvalidTarget);

    void
        setCurrentTemporalPosition(
            in TemporalPosition
                temporal_position)
    raises(InvalidTarget, InvalidParameter);

    void
        setSpeed(
            in Speed
                the_speed,
            in unsigned long
                transition_duration)
    raises(InvalidTarget, InvalidParameter);

    void
        setTimestones(
            in sequence<Timestone>
                timestones)
    raises(InvalidTarget, InvalidParameter);

    unsigned long
        getInitialTemporalPosition()
    raises(InvalidTarget);

    unsigned long
        getTerminalTemporalPosition()
    raises(InvalidTarget);

    unsigned long
        getVDLength(
            in GTIndicator
                gt_indicator)
    raises(InvalidTarget);

    TemporalTermination
        getTemporalTermination()
    raises(InvalidTarget);

    unsigned long
        getCurrentTemporalPosition()
    raises(InvalidTarget);

    short
        getSpeedRate()
    raises(InvalidTarget);

    unsigned long
        getOGTR()
    raises(InvalidTarget);

    short
        getEffectiveSpeedRate()
    raises(InvalidTarget);

    unsigned long
        getEffectiveOGTR()
    raises(InvalidTarget);

    unsigned short
        getTimestoneStatus()
    raises(InvalidTarget);
```

```
void
    setPerceptibleSizeProjection(
        in PerceptibleSizeProjection
            perceptible_size_projection,
        in unsigned long
            transition_duration)
raises(InvalidTarget, InvalidParameter);

void
    setAspectRatio(
        in AspectRatio
            preserved)
raises(InvalidTarget);

void
    setVisibleSize(
        in VSGS
            the_vsgs,
        in VS
            the_vs,
        in unsigned long
            transition_duration)
raises(InvalidTarget, InvalidParameter);

void
    setVisibleSizesAdjustment(
        in sequence<AdjustmentAxis>
            set_of_axes,
        in AdjustmentPolicy
            adjustment_policy,
        in unsigned long
            transition_duration)
raises(InvalidTarget);

void
    setBox(
        in BoxConstants
            box)
raises(InvalidTarget);

void
    setDefaultBackground(
        in unsigned short
            background,
        in unsigned long
            transition_duration)
raises(InvalidTarget, InvalidParameter);

void
    setAttachmentPoint(
        in AttachmentPointType
            type,
        in AttachmentPoint
            positions)
raises(InvalidTarget, InvalidParameter);

void
    setAttachmentPointPosition(
        in AttachmentPointType
            type,
        in ReferenceType
            vseap_reference_point,
        in Lengths
            the_lengths,
        in unsigned long
            transition_duration)
raises(InvalidTarget, InvalidParameter);

void
    setVisibleSizesAlignment(
        in SizeBorder
            size_border,
        in long
            interval,
        in unsigned long
            transition_duration)
raises(InvalidTarget);

void
    setMovingAbility(
        in UserControls
            moving_ability)
raises(InvalidTarget);
```

```
    void
        setResizingAbility(
            in UserControls
                resizing_ability)
    raises(InvalidTarget);

    void
        setScalingAbility(
            in UserControls
                scaling_ability)
    raises(InvalidTarget);

    void
        setScrollingAbility(
            in UserControls
                scrolling_ability)
    raises(InvalidTarget);

    unsigned short
        getGSR()
    raises(InvalidTarget);

    SpecifiedPosition
        getPS(
            in GSIndicator
                gs)
    raises(InvalidTarget);

    AspectRatio
        getAspectRatio()
    raises(InvalidTarget);

    SpecifiedPosition
        getPSAP(
            in PointType
                point_type)
    raises(InvalidTarget);

    VSGS
        getVSGS()
    raises(InvalidTarget);

    SpecifiedPosition
        getVS()
    raises(InvalidTarget);

    BoxConstants
        getBox()
    raises(InvalidTarget);

    unsigned short
        getDefaultBackground()
    raises(InvalidTarget);

    SpecifiedPosition
        getVSIAP(
            in PointType
                point_type)
    raises(InvalidTarget);

    SpecifiedPosition
        getVSIAPPosition()
    raises(InvalidTarget);

    SpecifiedPosition
        getVSEAP(
            in PointType
                point_type)
    raises(InvalidTarget);

    SpecifiedPosition
        getVSEAPPosition(
            in ReferencePoint
                reference_point)
    raises(InvalidTarget);

    UserControls
        getMovingAbility()
    raises(InvalidTarget);

    UserControls
        getResizingAbility()
    raises(InvalidTarget);

    UserControls
```

```
    getScalingAbility()
raises(InvalidTarget);

UserControls
    getScrollingAbility()
raises(InvalidTarget);

void
    setSelectability(
        in unsigned short
            min_number_of_selections,
        in unsigned short
            max_number_of_selections)
raises(InvalidTarget, InvalidParameter);

void
    setSelectionStatus(
        in SelectionStatusValue
            selection_state)
raises(InvalidTarget);

void
    setSelectionPresentationEffectResponsibility(
        in Responsibility
            the_responsibility)
raises(InvalidTarget);

void
    getSelectability(
        out unsigned short
            min_mumber_of_selections,
        out unsigned short
            max_mumber_of_selections)
raises(InvalidTarget);

EffectiveSelectability
    getEffectiveSelectability()
raises(InvalidTarget);

SelectionStatusValue
    getSelectionStatus()
raises(InvalidTarget);

SelectionModeValue
    getSelectionMode()
raises(InvalidTarget);

Responsibility
    getSelectionPresentationEffectResponsibility()
raises(InvalidTarget);

void
    setModifiability(
        in unsigned short
            min_number_of_modifications,
        in unsigned short
            max_number_of_modifications)
raises(InvalidTarget, InvalidParameter);

void
    setModificationStatus(
        in ModificationStatusValue
            modification_state)
raises(InvalidTarget);

void
    setModificationPresentationEffectResponsibility(
        in Responsibility
            the_responsibility)
raises(InvalidTarget);

void
    getModifiability(
        out unsigned short
            min_numbers_of_modifications,
        out unsigned short
            max_numbers_of_modifications)
raises(InvalidTarget);

EffectiveModifiability
    getEffectiveModifiability()
raises(InvalidTarget);

ModificationStatusValue
    getModificationStatus()
```

```
    raises(InvalidTarget);

    ModificationModeValue
        getModificationMode()
    raises(InvalidTarget);

    Responsibility
        getModificationPresentationEffectResponsibility()
    raises(InvalidTarget);

    void
        setNoInteractionStyle()
    raises(InvalidTarget);

};
```

7.1.22 `RtComponent` object

For the `RtComponent` object no specific operations are defined. The object inherits from the `RtComponentOrSocket` object and from the `RtObject` object .

7.1.22.1 IDL description

```
interface   RtComponent: RtComponentOrSocket, RtObject {};
```

7.1.23 `RtCompositeOrStructuralSocket` object

The following subclause defines the operations of the `RtCompositeOrStructuralSocket` object.

7.1.23.1 `setResizingStrategy` operation

Synopsis:

Interface:	`RtCompositeOrStructuralSocket`	
Operation:	`setResizingStrategy`	
Result:	`void`	
In:	`ResizingStrategy`	`resizing_strategy`
Exception:	`InvalidTarget`	

Description:

This operation specifies the PS resizing strategy that an rt-composite or structural socket is to have regarding the modification of the VSs of the child sockets having a Parent Relative Generic Space (PRGS).

The `setResizingStrategy` operation triggers the execution of the "set resizing strategy" elementary action with the bound rt-composite or structural socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 53.2.2.

The `resizing_strategy` parameter specifies the value of the "resizing strategy" parameter of the "set resizing strategy" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.23.2 `getResizingStrategy` operation

Synopsis:

Interface:	`RtCompositeOrStructuralSocket`
Operation:	`getResizingStrategy`
Result:	`ResizingStrategy`
Exception:	`InvalidTarget`

Description:

This operation retrieves the resizing strategy value of the rt-composite or structural socket.

The `getResizingStrategy` operation triggers the execution of the "get resizing strategy" elementary action with the bound rt-composite or structural socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 53.4.3.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.23.3 `setAudibleCompositionEffect` operation

Synopsis:

Interface:	`RtCompositeOrStructuralSocket`	
Operation:	`setAudibleCompositionEffect`	
Result:	`void`	
In:	`unsigned short`	`audible_effect`
In:	`unsigned long`	`transition_duration`
Exception:	`InvalidTarget`	

Description:

This operation specifies the audible composition effect of an rt-composite or a structural socket. This effect is to be propagated to their descendant sockets having a PRGS. It is used to calculate the effective Original Volume (OV) of the descendant sockets having a PRGS.

The `setAudibleCompositionEffect` operation triggers the execution of the "set audible composition effect" elementary action with the bound rt-composite or structural socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 54.2.2.

The `audible_effect` parameter specifies the value of the "audible effect" parameter of the "set audible composition effect" action.

The `transition_duration` parameter specifies the value of the "transition duration" parameter of the "set audible composition effect" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.23.4 `getAudibleCompositionEffect` operation

Synopsis:

Interface:	`RtCompositeOrStructuralSocket`
Operation:	`getAudibleCompositionEffect`
Result:	`unsigned short`
Exception:	`InvalidTarget`

Description:

This operation retrieves the audible composition effect value of the rt-composite or structural socket. This effect is expressed as a percentage and used to determine the effective OV of the child sockets of the rt-composite or structural socket having as PRGS the rt-composite or structural socket. This effect is recursive for the child sockets of the structural sockets having as PRGS the rt-composite or structural socket, and so on.

The `getAudibleCompositionEffect` operation triggers the execution of the "get audible composition effect" elementary action with the bound rt-composite or structural socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 54.3.3.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.23.5 `getNumberOfSelectedSockets` operation

Synopsis:

Interface: `RtCompositeOrStructuralSocket`
Operation: `getNumberOfSelectedSockets`
Result: `unsigned short`
Exception: `InvalidTarget`

Description:

This operation retrieves the "number of selected sockets" attribute value of the rt-composite or structural socket.

The `getNumberOfSelectedSockets` operation triggers the execution of the "get number of selected sockets" elementary action with the bound rt-composite or structural socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 57.3.5.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.23.6 `getNumberOfModifiedSockets` operation

Synopsis:

Interface: `RtCompositeOrStructuralSocket`
Operation: `getNumberOfModifiedSockets`
Result: `unsigned short`
Exception: `InvalidTarget`

Description:

This operation retrieves the "number of modified sockets" attribute value of the rt-composite or structural socket.

The `getNumberOfModifiedSockets` operation triggers the execution of the "get number of modified sockets" elementary action with the bound rt-composite or structural socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 58.3.5.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.23.7 `setMenuInteractionStyle` operation

Synopsis:

Interface:	`RtCompositeOrStructuralSocket`	
Operation:	`setMenuInteractionStyle`	
Result:	`void`	
In:	`Orientation`	`upper_menu_orientation`
In:	`sequence <Association>`	`list_of_associations`
Exception:	`InvalidTarget`	
Exception:	`InvalidParameter`	

Description:

This operation assigns the menu interaction style to an rt-composite or a structural socket. This operation defines a style which affects the complete rt-composite or structural socket, i.e. all generations.

The `setMenuInteractionStyle` operation triggers the execution of the "set menu style" elementary action with the bound rt-composite or structural socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 59.2.4.

The `upper_menu_orientation` parameter specifies the value of the "upper menu orientation" parameter of the "set menu style" action.

The `list_of_associations` parameter specifies the value of the "list of associations" parameter of the "set menu style" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

The `InvalidParameter` exception is raised when the value of one of the parameters prohibits the normal execution of the action. The `completion_status` member indicates whether the action was completed (with a default value assigned to the inadequate parameter) or not. The `parameter_number` member identifies the rank of the invalid parameter.

7.1.23.8 `setScrollingListInteractionStyle` operation

Synopsis:

Interface:	`RtCompositeOrStructuralSocket`	
Operation:	`setScrollingListInteractionStyle`	
Result:	`void`	
In:	`PerceptibleReference`	`background`
In:	`unsigned short`	`visible_items_number`
In:	`SocketTail`	`first_item`
In:	`Separator`	`the_separator`
In:	`Orientation`	`the_orientation`
In:	`SliderSide`	`slider_side`
In:	`PerceptibleReference`	`slider`
In:	`PerceptibleReference`	`slider_cursor`
In:	`PerceptibleReference`	`slider_background`
In:	`long`	`slider_min_value`
In:	`long`	`slider_max_value`
Exception:	`InvalidTarget`	
Exception:	`InvalidParameter`	

Description:

This operation assigns the scrolling list interaction style to an rt-composite or a structural socket. This operation defines a style which affects the first generation and only the child presentable sockets of the rt-composite or structural socket.

The `setScrollingListInteractionStyle` operation triggers the execution of the "set scrolling list style" elementary action with the bound rt-composite or structural socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 59.2.5.

The `background` parameter specifies the value of the "background" parameter of the "set scrolling list style" action.

The `visible_items_number` parameter specifies the value of the "number of visible items" parameter of the "set scrolling list style" action.

The `first_item` parameter specifies the value of the "first item" parameter of the "set scrolling list style" action.

The `the_separator` parameter specifies the value of the "separator" parameter of the "set scrolling list style" action.

The `the_orientation` parameter specifies the value of the "scrolling list orientation" parameter of the "set scrolling list style" action.

The `slider_side` parameter specifies the value of the "slider side" parameter of the "set scrolling list style" action.

The `slider`, `slider_cursor`, `slider_background`, `slider_min_value`, `slider_max_value` parameters specify the values of the parameters of the "set slider style" action embedded by the "set scrolling list style" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

The `InvalidParameter` exception is raised when the value of one of the parameters prohibits the normal execution of the action. The `completion_status` member indicates whether the action was completed (with a default value assigned to the inadequate parameter) or not. The `parameter_number` member identifies the rank of the invalid parameter.

7.1.23.9 IDL description

```
interface   RtCompositeOrStructuralSocket   {

    void
        setResizingStrategy(
            in ResizingStrategy
                resizing_strategy)
    raises(InvalidTarget);

    ResizingStrategy
        getResizingStrategy()
    raises(InvalidTarget);

    void
        setAudibleCompositionEffect(
            in unsigned short
                audible_effect,
            in unsigned long
                transition_duration)
    raises(InvalidTarget);

    unsigned short
        getAudibleCompositionEffect()
    raises(InvalidTarget);

    unsigned short
```

```
        getNumberOfSelectedSockets()
    raises(InvalidTarget);

    unsigned short
        getNumberOfModifiedSockets()
    raises(InvalidTarget);

    void
        setMenuInteractionStyle(
            in Orientation
                upper_menu_orientation,
            in sequence <Association>
                list_of_associations)
    raises(InvalidTarget, InvalidParameter);

    void
        setScrollingListInteractionStyle(
            in PerceptibleReference
                background,
            in unsigned short
                visible_items_number,
            in SocketTail
                first_item,
            in Separator
                the_separator,
            in Orientation
                the_orientation,
            in SliderSide
                slider_side,
            in PerceptibleReference
                slider,
            in PerceptibleReference
                slider_cursor,
            in PerceptibleReference
                slider_background,
            in long
                slider_min_value,
            in long
                slider_max_value)
    raises(InvalidTarget, InvalidParameter);

};
```

7.1.24 `RtComposite` object

For the `RtComposite` object no specific operations are defined. The object inherits from the `RtCompositeOrStructuralSocket` object and from the `RtComponent` object.

7.1.24.1 IDL description

```
interface   RtComposite: RtCompositeOrStructuralSocket, RtComponent {};
```

7.1.25 `StructuralSocket` object

For the `StructuralSocket` object no specific operations are defined. The object inherits from the `RtCompositeOrStructuralSocket` object and from the `Socket` object.

7.1.25.1 IDL description

```
interface   StructuralSocket: RtCompositeOrStructuralSocket, Socket {};
```

7.1.26 `RtGenericContentOrPresentableSocket` object

The following subclause defines the operations of the `RtGenericContentOrPresentableSocket` object.

7.1.26.1 `setAudibleVolume` operation

Synopsis:

Interface:	`RtGenericContentOrPresentableSocket`	
Operation:	`setAudibleVolume`	
Result:	`void`	
In:	`AudibleVolume`	`audible_volume`
In:	`unsigned long`	`transition_duration`
Exception:	`InvalidTarget`	
Exception:	`InvalidParameter`	

Description:

This action specifies the audible volume of an rt-content, an rt-multiplexed content, a presentable socket or a multiplexed presentable socket.

The `setAudibleVolume` operation triggers the execution of the "set audible volume" elementary action with the bound rt-content, rt-multiplexed content, presentable socket or multiplexed presentable socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 54.2.1.

The `audible_volume` parameter specifies the value of the "audible volume" parameter of the "set audible volume" action.

The `transition_duration` parameter specifies the value of the "transition duration" parameter of the "set audible volume" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

The `InvalidParameter` exception is raised when the value of one of the parameters prohibits the normal execution of the action. The `completion_status` member indicates whether the action was completed (with a default value assigned to the inadequate parameter) or not. The `parameter_number` member identifies the rank of the invalid parameter.

7.1.26.2 `getInitialOriginalAudibleVolume` operation

Synopsis:

Interface:	`RtGenericContentOrPresentableSocket`
Operation:	`getInitialOriginalAudibleVolume`
Result:	`unsigned long`
Exception:	`InvalidTarget`

Description:

This operation retrieves the initial original audible volume value of the rt-content, rt-multiplexed content, presentable socket or multiplexed presentable socket. This initial volume is expressed in original generic audible volume unit within the interval defined by the original audible volume range.

The `getInitialOriginalAudibleVolume` operation triggers the execution of the "get IOV" elementary action with the bound rt-content, rt-multiplexed content, presentable socket or multiplexed presentable socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 54.3.1.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.26.3 `getCurrentOriginalAudibleVolume` operation

Synopsis:

Interface:	`RtGenericContentOrPresentableSocket`
Operation:	`getCurrentOriginalAudibleVolume`
Result:	`unsigned long`
Exception:	`InvalidTarget`

Description:

This operation retrieves the current original audible volume value of the rt-content, rt-multiplexed content, presentable socket or multiplexed presentable socket. This current volume is expressed in original generic audible volume unit within the interval defined by the original audible volume range.

The `getCurrentOriginalAudibleVolume` operation triggers the execution of the "get current OV" elementary action with the bound rt-content, rt-multiplexed content, presentable socket or multiplexed presentable socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 54.3.2.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.26.4 `getEffectiveOriginalAudibleVolume` operation

Synopsis:

Interface:	`RtGenericContentOrPresentableSocket`
Operation:	`getEffectiveOriginalAudibleVolume`
Result:	`unsigned long`
Exception:	`InvalidTarget`

Description:

This operation retrieves the effective original audible volume value of the rt-content, rt-multiplexed content, presentable socket or multiplexed presentable socket. This effective volume is expressed in original generic audible volume unit within the interval defined by the original audible volume range. It is calculated by the MHEG engine using the current original audible volume and the audible composition effect.

The `getEffectiveOriginalAudibleVolume` operation triggers the execution of the "get effective OV" elementary action with the bound rt-content, rt-multiplexed content, presentable socket or multiplexed presentable socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 54.3.4.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.26.5 `getPerceptibleAudibleVolume` operation

Synopsis:

Interface:	`RtGenericContentOrPresentableSocket`
Operation:	`getPerceptibleAudibleVolume`
Result:	`unsigned long`
Exception:	`InvalidTarget`

Description:

This operation retrieves the perceptible original audible volume value of the rt-content, rt-multiplexed content, presentable socket or multiplexed presentable socket in the assigned channel. This perceptible volume is expressed in channel generic audible volume unit within the interval defined by the channel audible volume range. It is calculated by the MHEG engine and corresponds to a projection of the effective original audible volume in the channel generic space.

The `getPerceptibleAudibleVolume` operation triggers the execution of the "get perceptible OV" elementary action with the bound rt-content, rt-multiplexed content, presentable socket or multiplexed presentable socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 54.3.5.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.26.6 `setButtonInteractionStyle` operation

Synopsis:

Interface:	`RtGenericContentOrPresentableSocket`	
Operation:	`setButtonInteractionStyle`	
Result:	`void`	
In:	`PresentationState`	`initial_state`
In:	`AlternatePresentation`	`alternate_presentation_1`
In:	`AlternatePresentation`	`alternate_presentation_2`
In:	`AlternatePresentation`	`alternate_presentation_3`
Exception:	`InvalidTarget`	
Exception:	`InvalidParameter`	

Description:

This operation assigns the button interaction style to an rt-content, an rt-multiplexed content, a presentable socket or a multiplexed presentable socket.

The `setButtonInteractionStyle` operation triggers the execution of the "set button style" elementary action with the bound rt-content, rt-multiplexed content, presentable socket or multiplexed presentable socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 59.2.1.

The `initial_state` parameter specifies the value of the "initial state" parameter of the "set button style" action.

The `alternate_presentation_1` parameter specifies the value of the "alternate presentation 1" parameter of the "set button style" action.

The `alternate_presentation_2` parameter specifies the value of the "alternate presentation 2" parameter of the "set button style" action.

The alternate_presentation_3 parameter specifies the value of the "alternate presentation 3" parameter of the "set button interaction style" action.

The InvalidTarget exception is raised when the object instance does not represent a valid target for the normal completion of the action. The period member returns the current period of the target.

The InvalidParameter exception is raised when the value of one of the parameters prohibits the normal execution of the action. The completion_status member indicates whether the action was completed (with a default value assigned to the inadequate parameter) or not. The parameter_number member identifies the rank of the invalid parameter.

7.1.26.7 IDL description

```
interface   RtGenericContentOrPresentableSocket {

    void
        setAudibleVolume(
            in AudibleVolume
                audible_volume,
            in unsigned long
                transition_duration)
    raises(InvalidTarget, InvalidParameter);

    unsigned long
        getInitialOriginalAudibleVolume()
    raises(InvalidTarget);

    unsigned long
        getCurrentOriginalAudibleVolume()
    raises(InvalidTarget);

    unsigned long
        getEffectiveOriginalAudibleVolume()
    raises(InvalidTarget);

    unsigned long
        getPerceptibleAudibleVolume()
    raises(InvalidTarget);

    void
        setButtonInteractionStyle(
            in PresentationState
                initial_state,
            in AlternatePresentation
                alternate_presentation_1,
            in AlternatePresentation
                alternate_presentation_2,
            in AlternatePresentation
                alternate_presentation_3)
    raises(InvalidTarget, InvalidParameter);

};
```

7.1.27 RtGenericContent object

For the RtGenericContent object no specific operations are defined. The object inherits from the RtGenericContentOrPresentableSocket object and from the RtComponent object.

7.1.27.1 IDL description

```
interface   RtGenericContent: RtGenericContentOrPresentableSocket, RtComponent {};
```

7.1.28 GenericPresentableSocket object

For the GenericPresentableSocket object no specific operations are defined. The object inherits from the RtGenericContentOrPresentableSocket object and from the Socket object.

7.1.28.1 IDL description

```
interface   GenericPresentableSocket: RtGenericContentOrPresentableSocket, Socket {};
```

7.1.29 `RtContentOrPresentableSocket` object

The following subclause defines the operations of the `RtContentOrPresentableSocket` object.

7.1.29.1 `setSliderInteractionStyle` operation

Synopsis:

Interface:	`RtContentOrPresentableSocket`	
Operation:	`setSliderInteractionStyle`	
Result:	`void`	
In:	`PerceptibleReference`	`cursor`
In:	`PerceptibleReference`	`background`
In:	`Orientation`	`the_orientation`
In:	`short`	`min_value`
In:	`short`	`max_value`
Exception:	`InvalidTarget`	
Exception:	`InvalidParameter`	

Description:

This operation assigns the slider interaction style to an rt-content or a presentable socket created from a content object model which contains a generic numeric as "content data".

The `setSliderStyle` operation triggers the execution of the "set slider style" elementary action with the bound rt-content or presentable socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 59.2.2.

The `cursor` parameter specifies the value of the "cursor" parameter of the "set slider style" action.

The `background` parameter specifies the value of the "background" parameter of the "set slider style" action.

The `the_orientation` parameter specifies the value of the "orientation" parameter of the "set slider style" action.

The `min_value` parameter specifies the value of the "minimum value" parameter of the "set slider style" action.

The `max_value` parameter specifies the value of the "maximum value" parameter of the "set slider style" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

The `InvalidParameter` exception is raised when the value of one of the parameters prohibits the normal execution of the action. The `completion_status` member indicates whether the action was completed (with a default value assigned to the inadequate parameter) or not. The `parameter_number` member identifies the rank of the invalid parameter.

7.1.29.2 `setEntryFieldInteractionStyle` operation

Synopsis:

Interface:	`RtContentOrPresentableSocket`	
Operation:	`setEntryFieldInteractionStyle`	
Result:	`void`	
In:	`EchoStyle`	`echo_style`
In:	`PerceptibleReference`	`background`
Exception:	`InvalidTarget`	
Exception:	`InvalidParameter`	

Description:

This operation assigns the entry field interaction style to an rt-content or a presentable socket created from a content object model which contains a generic numeric or a generic string as "content data".

The `setEntryFieldInteractionStyle` operation triggers the execution of the "set entry field style" elementary action with the bound rt-content or presentable socket as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 59.2.3.

The `echo_style` parameter specifies the value of the "echo style" parameter of the "set entry field style" action.

The `background` parameter specifies the value of the "background" parameter of the "set entry field style" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

The `InvalidParameter` exception is raised when the value of one of the parameters prohibits the normal execution of the action. The `completion_status` member indicates whether the action was completed (with a default value assigned to the inadequate parameter) or not. The `parameter_number` member identifies the rank of the invalid parameter.

7.1.29.3 IDL description

```
interface   RtContentOrPresentableSocket    {

    void
        setSliderInteractionStyle(
            in PerceptibleReference
                cursor,
            in PerceptibleReference
                background,
            in Orientation
                the_orientation,
            in short
                min_value,
            in short
                max_value)
    raises(InvalidTarget, InvalidParameter);

    void
        setEntryFieldInteractionStyle(
            in EchoStyle
                echo_style,
            in PerceptibleReference
                background)
    raises(InvalidTarget, InvalidParameter);

};
```

7.1.30 `RtContent` object

For the `RtContent` object no specific operations are defined. The object inherits from the `RtContentOrPresentableSocket` object and from the `RtGenericContent` object.

7.1.30.1 IDL description

```
interface   RtContent: RtContentOrPresentableSocket, RtGenericContent {};
```

7.1.31 `PresentableSocket` object

For the `PresentableSocket` object no specific operations are defined. The object inherits from the `RtContentOrPresentableSocket` object and from the `GenericPresentableSocket` object.

7.1.31.1 IDL description

```
interface   PresentableSocket: RtContentOrPresentableSocket, GenericPresentableSocket {};
```

7.1.32 `RtMultiplexedContentOrPresentableSocket` object

The following subclause defines the operations of the `RtMultiplexedContentOrPresentableSocket` object.

7.1.32.1 `setStreamChoice` operation

Synopsis:

Interface:	`RtMultiplexedContentOrPresentableSocket`	
Operation:	`setStreamChoice`	
Result:	`void`	
In:	`StreamIdentifier`	`stream_identifier`
Exception:	`InvalidTarget`	
Exception:	`InvalidParameter`	

Description:

This operation specifies a stream to be chosen in the multiplexed data and assigned to the rt-multiplexed content or multiplexed presentable socket. Once a stream is chosen for an rt-multiplexed content or a multiplexed presentable socket, when it becomes running, the rt-multiplexed content or the multiplexed presentable socket is responsible for the presentation of this chosen stream.

The `setStreamChoice` operation triggers the execution of the "set stream choice" elementary action with the bound rt-multiplexed content or multiplexed presentable socket as its single target.

The `stream_identifier` parameter specifies the value of the "stream choice" parameter of the "set stream choice" action.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 55.2.1.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

The `InvalidParameter` exception is raised when the value of one of the parameters prohibits the normal execution of the action. The `completion_status` member indicates whether the action was completed (with a default value assigned to the inadequate parameter) or not. The `parameter_number` member identifies the rank of the invalid parameter.

7.1.32.2 `getStreamChosen` operation

Synopsis:

Interface: `RtMultiplexedContentOrPresentableSocket`
Operation: `getStreamChosen`
Result: `StreamValue`
Exception: `InvalidTarget`

Description:

This operation retrieves the stream chosen for the rt-multiplexed content or multiplexed presentable socket.

The `getStreamChosen` operation triggers the execution of the "get stream chosen" elementary action with the bound rt-multiplexed content or multiplexed presentable socket as its single target.

The effect of the action on its target, the semantics of its parameters, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 55.3.1.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.32.3 IDL description

```
interface   RtMultiplexedContentOrPresentableSocket {

    void
        setStreamChoice(
            in StreamIdentifier
                stream_identifier)
    raises(InvalidTarget, InvalidParameter);

    StreamValue
        getStreamChosen()
    raises(InvalidTarget);
};
```

7.1.33 `RtMultiplexedContent` object

For the `RtMultiplexedContent` object no specific operations are defined. The object inherits from the `RtMultiplexedContentOrPresentableSocket` object and from the `RtGenericContent` object.

7.1.33.1 IDL description

```
interface   RtMultiplexedContent: RtMultiplexedContentOrPresentableSocket, RtGenericContent {};
```

7.1.34 `MultiplexedPresentableSocket` object

For the `MultiplexedPresentableSocket` object no specific operations are defined. The object inherits from the `RtMultiplexedContentOrPresentableSocket` object and from the `GenericPresentableSocket` object.

7.1.34.1 IDL description

```
interface   MultiplexedPresentableSocket: RtMultiplexedContentOrPresentableSocket,
GenericPresentableSocket {};
```

7.1.35 `Channel` object

The following subclause defines the operations of the `Channel` object. The object inherits from the `Entity` object.

7.1.35.1 `bind` operation

Synopsis:

Interface:	`Channel`	
Operation:	`bind`	
Result:	`ChannelIdentifier`	
In:	`ChannelReference`	`channel_reference`
Exception:	`AlreadyBound`	
Exception:	`InvalidTarget`	

Description:

This operation binds the Channel instance (an interface object instance) with a channel (an MHEG entity).

The `channel_reference` parameter specifies the reference of the channel.

The operation returns the identifier of the bound channel.

The `AlreadyBound` exception is raised when the interface object instance is already bound with an MHEG entity.

The `InvalidTarget` exception is raised when the targeted MHEG entity is not available. The `period` member returns the current period of the target.

7.1.35.2 `unbind` operation

Synopsis:

Interface:	`Channel`
Operation:	`unbind`
Result:	`void`
Exception:	`NotBound`

Description:

This operation cancels the binding between the Channel instance (an interface object instance) and a channel (an MHEG entity).

The `NotBound` exception is raised when the interface object instance is not bound with an MHEG entity.

7.1.35.3 `new` operation

Synopsis:

Interface:	`Channel`	
Operation:	`new`	
Result:	`ChannelIdentifier`	
In:	`ChannelReference`	`channel_reference`
In:	`OriginalDefDeclaration`	`original_definition_declaration`
Exception:	`AlreadyBound`	
Exception:	`InvalidTarget`	

Description:

This operation enables the creation of a channel by the MHEG engine.

The `new` operation triggers the execution of the "new channel" elementary action targeted at a single channel.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 61.2.1.

The `channel_reference` parameter specifies a reference to a channel.

The `original_definition_declaration` parameter specifies the value of the "original definition declaration" parameter of the "new channel" action.

This operation implicitly binds the Channel instance (an interface object instance) with the new created channel (an MHEG entity).

The operation returns the identifier of the new created channel bound with the Channel instance.

The `AlreadyBound` exception is raised when the interface object instance is already bound with an MHEG entity.

The `InvalidTarget` exception is raised when the targeted MHEG entity is not available. The `period` member returns the current period of the target.

7.1.35.4 `delete` operation

Synopsis:

Interface:	`Channel`
Operation:	`delete`
Result:	`void`
Exception:	`NotBound`
Exception:	`InvalidTarget`

Description:

This operation enables the removing of a channel by the MHEG engine.

The `delete` operation triggers the execution of the "delete channel" elementary action targeted at a single channel.

The effect of the action on its target and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 61.2.2.

This operation implicitly cancels the binding between the Channel instance (an interface object instance) and the new deleted channel (an MHEG entity).

The `NotBound` exception is raised when the interface object instance is not bound with an MHEG entity.

The `InvalidTarget` exception is raised when the targeted MHEG entity is not available. The `period` member returns the current period of the target.

7.1.35.5 `getRtAvailabilityStatus` operation

Synopsis:

Interface:	`Channel`
Operation:	`getAvailability`
Result:	`ChannelStatusValue`
Exception:	`NotBound`
Exception:	`InvalidTarget`

Description:

This operation retrieves the availability of a channel to the MHEG engine.

The `getAvailability` operation triggers the execution of the "get channel availability status" elementary action with the bound channel as its single target.

The effect of the action on its target, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 61.3.1.

The operation returns the availability of the channel bound with the Channel instance. The returned value is either `NOT_AVAILABLE`, `PROCESSING` or `AVAILABLE`.

When the returned value is `NOT_AVAILABLE`, the operation implicitly cancels the binding between the Channel instance (an interface object instance) and the channel (an MHEG entity).

The `NotBound` exception is raised when the interface object instance is not bound with an MHEG entity.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.35.6 `getIdentifier` operation

Synopsis:

Interface:	`Channel`
Operation:	`getIdentifier`
Result:	`ChannelIdentifier`
Exception:	`NotBound`

Description:

This operation retrieves the identifier of the channel (an MHEG entity) bound with the Channel instance (an interface object instance).

The `NotBound` exception is raised when the interface object instance is not bound with an MHEG entity.

7.1.35.7 `kill` operation

Synopsis:

Interface:	`Channel`
Operation:	`kill`
Result:	`void`

Description:

This operation deletes the Channel instance (an interface object instance).

7.1.35.8 `setPerceptability` operation

Synopsis:

Interface:	`Channel`	
Operation:	`setPerceptability`	
Result:	`void`	
In:	`ChannelPerceptabilityValue`	`channel_perceptability`
Exception:	`InvalidTarget`	

Description:

This operation enables a channel to be turned on or off. This is used to enable or disable the perception of a channel by a user.

The `setPerceptability` operation triggers the execution of the "set channel perceptability" elementary action with the bound channel as its single target.

The effect of the action on its target, the semantics of its parameters and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 62.2.1.

The `channel_perceptability` parameter specifies the value of the "perceptability" parameter of the "set channel perceptability" action.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.35.9 `getPerceptability` operation

Synopsis:

Interface: `Channel`
Operation: `getPerceptability`
Result: `ChannelPerceptabilityValue`
Exception: `InvalidTarget`

Description:

This operation retrieves the perceptability of a channel.

The `getPerceptability` operation triggers the execution of the "get channel perceptability" elementary action with the bound channel as its single target.

The effect of the action on its target, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC DIS 13522-1 [1], subclause 62.3.1.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.35.10 `getAssignedPerceptibles` operation

Synopsis:

Interface: `Channel`
Operation: `getAssignedPerceptibles`
Result: `sequence<PerceptibleReference>`
Exception: `InvalidTarget`

Description:

This operation retrieves the perceptibles assigned to the channel.

The `getAssignedPerceptibles` operation has no corresponding MHEG elementary action. It is symmetrical to the "get RGS" elementary action which retrieves the channel assigned to a perceptible.

The `InvalidTarget` exception is raised when the object instance does not represent a valid target for the normal completion of the action. The `period` member returns the current period of the target.

7.1.35.11 IDL description

```
interface   Channel: Entity {

    ChannelIdentifier
        bind(
            in ChannelReference
                channel_reference)
    raises(AlreadyBound, InvalidTarget);

    void
        unbind()
    raises(NotBound);

    ChannelIdentifier
        new(
            in ChannelReference
                channel_reference,
            in OriginalDefDeclaration
                original_definition_declaration)
```

```
    raises(AlreadyBound, InvalidTarget);

    void
        delete()
    raises(NotBound, InvalidTarget);

    ChannelStatusValue
        getAvailability()
    raises(NotBound, InvalidTarget);

    ChannelIdentifier
        getIdentifier()
    raises(NotBound);

    void
        kill();

    void
        setPerceptability(
            in ChannelPerceptabilityValue
                channel_perceptability)
    raises(InvalidTarget);

    ChannelPerceptabilityValue
        getPerceptability()
    raises(InvalidTarget);

    sequence<PerceptibleReference>
        getAssignedPerceptibles()
    raises(InvalidTarget);

};
```

7.1.36 Parameter definition

The following subclause defines the parameters that are used by the mandatory primitives.

```
//=====================================================================
typedef sequence<long> ApplicationIdentifier;


// Corresponding MHEG datatype: Object-Number
//=====================================================================
typedef long ObjectNumber;


// Interface: MhObject Operation: bind
// Interface: MhObject Operation: prepare
// Interface: MhObject Operation: getIdentifier
// Corresponding MHEG datatype: MHEG-Identifier
//=====================================================================
struct MHEGIdentifier {
    sequence<ApplicationIdentifier,1>
        application_identifier;
    ObjectNumber
        object_number;
};


// Corresponding MHEG datatype: Public-Identifier
//=====================================================================
typedef string PublicIdentifier;


// Corresponding MHEG datatype: System-Identifier
//=====================================================================
typedef string SystemIdentifier;


// Corresponding MHEG datatype: External-Long-Identifier
//=====================================================================
struct ExternalLongIdentifier {
    PublicIdentifier
        public_identifier;
    SystemIdentifier
        system_identifier;
};


// Corresponding MHEG datatype: Alias
//=====================================================================
typedef string Alias;
```

```
// Corresponding MHEG datatype: Container-Child-Reference
//=====================================================================
enum ContainerChildReference {
    CHILD,
    DESCENDANT
};


// Interface: MhObject Operation: getPreparationStatus
// Corresponding MHEG datatype: Preparation-Status-Value
//=====================================================================
enum PreparationStatusValue {
    READY,
    NOT_READY,
    PROCESSING
};


// Interface: MhMultiplexedContent Operation: setMultiplex
// Interface: MhMultiplexedContent Operation: setDemultiplex
// Interface: RtMultiplexedContentOrPresentableSocket Operation: setStreamChoice
// Corresponding MHEG datatype: Stream-Identifier
//=====================================================================
typedef sequence<long> StreamIdentifier;


// Corresponding MHEG datatype: Rt-Dynamic-Reference
//=====================================================================
enum RtDynamicReference {
    QUESTION_MARK,
    STAR
};


// Interface: RtObject Operation: getAvailabilityStatus
// Corresponding MHEG datatype: Rt-Availibility-Status-Value
//=====================================================================
enum RtAvailabilityStatusValue {
    RT_AVAILIBILITY_STATUS_VALUE_AVAILABLE,
    RT_AVAILIBILITY_STATUS_VALUE_NOT_AVAILABLE,
    RT_AVAILIBILITY_STATUS_VALUE_PROCESSING
};


// Interface: RtObject Operation: getRunningStatus
// Corresponding MHEG datatype: Running-Status-Value
//=====================================================================
enum RunningStatusValue {
    RUNNING_STATUS_VALUE_RUNNING,
    RUNNING_STATUS_VALUE_NOT_RUNNING,
    RUNNING_STATUS_VALUE_PROCESSING
};


// Interface: RtScript Operation: getTerminationStatus
// Corresponding MHEG datatype: Termination-Status-Value
//=====================================================================
enum TerminationStatusValue {
    TERMINATED,
    NOT_TERMINATED
};


// Interface: RtComponentOrSocket Operation: setRGS
// Interface: Channel Operation: getIdentifier
// Corresponding MHEG datatype: Channel-Identifier
//=====================================================================
typedef long ChannelIdentifier;


// Corresponding MHEG datatype: Priority-Level
//=====================================================================
enum PriorityLevel {
    INCREMENT_PRIORITY,
    DECREMENT_PRIORITY
};


// Interface: RtComponentOrSocket Operation: setVisibleDuration
// Corresponding MHEG datatype: Temporal-Position
//=====================================================================
enum TemporalPositionTag { SPECIFIED_TEMPORAL_POINT_TAG, LOGICAL_TEMPORAL_PD_POINT_TAG };
union TemporalPosition
```

```
switch (TemporalPositionTag){
    case SPECIFIED_TEMPORAL_POINT_TAG:
        long
            specified_temporal_point;
    case LOGICAL_TEMPORAL_PD_POINT_TAG:
        long
            logical_temporal_PD_point;
};


// Interface: RtComponentOrSocket Operation: setCurrentTemporalPosition
// Corresponding MHEG datatype: Current-Temporal-Position
//=====================================================================
enum CurrentTemporalPositionTag { CURRENT_TEMPORAL_POSITION_SPECIFIED_TEMPORAL_POINT_TAG,
CURRENT_TEMPORAL_POSITION_LOGICAL_TEMPORAL_VD_POINT_TAG };
union CurrentTemporalPosition
switch (CurrentTemporalPositionTag){
    case CURRENT_TEMPORAL_POSITION_SPECIFIED_TEMPORAL_POINT_TAG:
        long
            specified_temporal_point;
    case CURRENT_TEMPORAL_POSITION_LOGICAL_TEMPORAL_VD_POINT_TAG:
        long
            logical_temporal_vd_point;
};


// Interface: RtComponentOrSocket Operation: setTemporalTermination
// Interface: RtComponentOrSocket Operation: getTemporalTermination
// Corresponding MHEG datatype: Temporal-Termination
//=====================================================================
enum TemporalTermination {
    TEMPORAL_TERMINATION_FREEZE,
    TEMPORAL_TERMINATION_STOP
};


// Interface: RtComponentOrSocket Operation: setSpeed
// Corresponding MHEG datatype: Speed
//=====================================================================
enum SpeedTag { SPECIFIED_OGTR_TAG, SPEED_RATE_TAG, SCALING_FACTOR_TAG };
union Speed
switch (SpeedTag){
    case SPECIFIED_OGTR_TAG:
        long
            specified_OGTR;
    case SPEED_RATE_TAG:
        long
            speed_rate;
    case SCALING_FACTOR_TAG:
        long
            scaling_factor;
};


// Corresponding MHEG datatype: Timestone-Position
//=====================================================================
enum TimestonePositionTag { TIMESTONE_POSITION_SPECIFIED_TEMPORAL_POINT_TAG,
TIMESTONE_POSITION_LOGICAL_TEMPORAL_PD_POINT_TAG,
TIMESTONE_POSITION_LOGICAL_TEMPORAL_VD_POINT_TAG };
union TimestonePosition
switch (TimestonePositionTag){
    case TIMESTONE_POSITION_SPECIFIED_TEMPORAL_POINT_TAG:
        long
            specified_temporal_point;
    case TIMESTONE_POSITION_LOGICAL_TEMPORAL_PD_POINT_TAG:
        long
            logical_temporal_PD_point;
    case TIMESTONE_POSITION_LOGICAL_TEMPORAL_VD_POINT_TAG:
        long
            logical_temporal_VD_point;
};


// Interface: RtComponentOrSocket Operation: getVDLength
// Corresponding MHEG datatype: GT-Indicator
//=====================================================================
enum GTIndicator {
    OGTU,
    RGTU
};


// Corresponding MHEG datatype: Perceptible-Projection
//=====================================================================
enum PerceptibleProjectionTag { SPECIFIED_SIZE_TAG, IOGSR_SCALING_FACTOR_TAG,
```

```
COGSR_SCALING_FACTOR_TAG };
union PerceptibleProjection
switch (PerceptibleProjectionTag){
    case SPECIFIED_SIZE_TAG:
        long
            specified_size;
    case IOGSR_SCALING_FACTOR_TAG:
        long
            iogsr_scaling_factor;
    case COGSR_SCALING_FACTOR_TAG:
        long
            cogsr_scaling_factor;
};


// Interface: RtComponentOrSocket Operation: setAspectRatioPreserved
// Interface: RtComponentOrSocket Operation: getAspectRatio
// Corresponding MHEG datatype: Aspect-Ratio
//=========================================================================
enum AspectRatio {
    PRESERVED,
    NOT_PRESERVED
};


// Interface: RtComponentOrSocket Operation: setVisibleSize
// Interface: RtComponentOrSocket Operation: getVSGS
// Corresponding MHEG datatype: VSGS
//=========================================================================
enum VSGS {
    THIS,
    RELATIVE
};


// Corresponding MHEG datatype: Size-Attribute
//=========================================================================
enum SizeAttributeTag { SIZE_ATTRIBUTE_SPECIFIED_SIZE_TAG, SIZE_ATTRIBUTE_IVS_RELATIVE_TAG,
SIZE_ATTRIBUTE_CVS_RELATIVE_TAG };
union SizeAttribute
switch (SizeAttributeTag){
    case SIZE_ATTRIBUTE_SPECIFIED_SIZE_TAG:
        long
            specified_size;
    case SIZE_ATTRIBUTE_IVS_RELATIVE_TAG:
        long
            ivs_relative;
    case SIZE_ATTRIBUTE_CVS_RELATIVE_TAG:
        long
            cvs_relative;
};


// Interface: RtComponentOrSocket Operation: setVisibleSizesAdjustment
// Corresponding MHEG datatype: Adjustment-Axis
//=========================================================================
enum AdjustmentAxis {
    X_AXIS,
    Y_AXIS,
    Z_AXIS
};


// Corresponding MHEG datatype: Sub-Socket-Reference
//=========================================================================
enum SubSocketReference {
    SUB_SOCKET_REFERENCE_CHILD,
    SUB_SOCKET_REFERENCE_DESCENDANT,
    SUB_SOCKET_REFERENCE_QUESTION_MARK_CHILD,
    SUB_SOCKET_REFERENCE_QUESTION_MARK_DESCENDANT
};


// Interface: RtComponentOrSocket Operation: setBox
// Interface: RtComponentOrSocket Operation: getBox
// Corresponding MHEG datatype: Box-Constants
//=========================================================================
enum BoxConstants {
    PRESENTED,
    NOT_PRESENTED
};


// Interface: RtComponentOrSocket Operation: setAttachmentPointPosition
// Corresponding MHEG datatype: Reference-Type
```

```
//=========================================================================
enum ReferenceType {
    VSIAP,
    VSEAP
};


// Interface: RtComponentOrSocket Operation: setAttachmentPoint
// Interface: RtComponentOrSocket Operation: setAttachmentPointPosition
// Corresponding MHEG datatype: Attachment-Point-Type
//=========================================================================
enum AttachmentPointType {
    ATTACHMENT_POINT_TYPE_PSAP,
    ATTACHMENT_POINT_TYPE_VSIAP,
    ATTACHMENT_POINT_TYPE_VSEAP
};

// Interface: RtComponentOrSocket Operation: setVisibleSizesAlignment
// Corresponding MHEG datatype: Size-Border
//=========================================================================
enum SizeBorder {
    TOP,
    BOTTOM,
    RIGHT,
    LEFT,
    UPPER_Z,
    LOWER_Z,
    CENTER_X,
    CENTER_Y,
    CENTER_Z
};


// Interface: RtComponentOrSocket Operation: setMovingAbility
// Interface: RtComponentOrSocket Operation: setResizingAbility
// Interface: RtComponentOrSocket Operation: setScalingAbility
// Interface: RtComponentOrSocket Operation: setScrollingAbility
// Interface: RtComponentOrSocket Operation: getMovingAbility
// Interface: RtComponentOrSocket Operation: getResizingAbility
// Interface: RtComponentOrSocket Operation: getScalingAbility
// Interface: RtComponentOrSocket Operation: getScrollingAbility
// Corresponding MHEG datatype: User-Controls
//=========================================================================
enum UserControls {
    ALLOWED,
    NOT_ALLOWED
};


// Interface: RtComponentOrSocket Operation: getPS
// Corresponding MHEG datatype: GS-Indicator
//=========================================================================
enum GSIndicator {
    OGSU,
    RGSU
};


// Interface: RtComponentOrSocket Operation: getPSAP
// Interface: RtComponentOrSocket Operation: getVSIAP
// Corresponding MHEG datatype: Point-Type
//=========================================================================
enum PointType {
    RELATIVE_POINT,
    ABSOLUTE_POINT
};


// Interface: RtComponentOrSocket Operation: setSelectionStatus
// Interface: RtComponentOrSocket Operation: getSelectionStatus
// Corresponding MHEG datatype: Selection-Status-Value
//=========================================================================
enum SelectionStatusValue {
    SELECTED,
    NOT_SELECTED
};


// Interface: RtComponentOrSocket Operation: setSelectionPresentationEffectResponsibility
// Interface: RtComponentOrSocket Operation: getSelectionPresentationEffectResponsibility
// Interface: RtComponentOrSocket Operation: setModificationPresentationEffectResponsibility
// Interface: RtComponentOrSocket Operation: getModificationPresentationEffectResponsibility
// Corresponding MHEG datatype: Responsibility
//=========================================================================
enum Responsibility {
```

```
    MHEG_ENGINE,
    AUTHOR
};


// Interface: RtComponentOrSocket Operation: getEffectiveSelectability
// Corresponding MHEG datatype: Effective-Selectability
//=========================================================================
enum EffectiveSelectability {
    EFFECTIVELY_SELECTABLE,
    EFFECTIVELY_NOT_SELECTABLE
};


// Interface: RtComponentOrSocket Operation: setModificationStatus
// Interface: RtComponentOrSocket Operation: getModificationStatus
// Corresponding MHEG datatype: Modification-Status-Value
//=========================================================================
enum ModificationStatusValue {
    MODIFIED,
    MODIFYING,
    NOT_MODIFIED
};


// Interface: RtComponentOrSocket Operation: getEffectiveModifiability
// Corresponding MHEG datatype: Effective-Modifiability
//=========================================================================
enum EffectiveModifiability {
    EFFECTIVELY_MODIFIABLE,
    EFFECTIVELY_NOT_MODIFIABLE
};


// Interface: RtCompositeOrStructuralSocket Operation: setResizingStrategy
// Interface: RtCompositeOrStructuralSocket Operation: getResizingStrategy
// Corresponding MHEG datatype: Resizing-Strategy
//=========================================================================
enum ResizingStrategy {
    FIXED,
    MINIMUM,
    GROWS_ONLY
};


// Interface: RtCompositeOrStructuralSocket Operation: setMenuInteractionStyle
// Interface: RtCompositeOrStructuralSocket Operation: setScrollingListInteractionStyle
// Interface: RtContentOrPresentableSocket Operation: setSliderInteractionStyle
// Corresponding MHEG datatype: Orientation
//=========================================================================
enum Orientation {
    HORIZONTAL,
    VERTICAL
};


// Corresponding MHEG datatype: Presentation-Persistence
//=========================================================================
enum PresentationPersistence {
    PERSISTENT,
    NOT_PERSISTENT
};


// Interface: RtCompositeOrStructuralSocket Operation: setScrollingListInteractionStyle
// Corresponding MHEG datatype: Slider-Side
//=========================================================================
enum SliderSide {
    SIDE1,
    SIDE2
};


// Interface: RtGenericContentOrPresentableSocket Operation: setAudibleVolume
// Corresponding MHEG datatype: Audible-Volume
//=========================================================================
enum AudibleVolumeTag { SPECIFIED_VOLUME_TAG, LOGICAL_VOLUME_TAG, IOV_SCALING_FACTOR_TAG,
OV_SCALING_FACTOR_TAG };
union AudibleVolume
switch (AudibleVolumeTag){
    case SPECIFIED_VOLUME_TAG:
        long
            specified_volume;
    case LOGICAL_VOLUME_TAG:
        long
```

```
            logical_volume;
    case IOV_SCALING_FACTOR_TAG:
        long
            iov_scaling_factor;
    case OV_SCALING_FACTOR_TAG:
        long
            ov_scaling_factor;
};


// Interface: RtGenericContentOrPresentableSocket Operation: setButtonInteractionStyle
// Corresponding MHEG datatype: Presentation-State
//=========================================================================
enum PresentationState {
    SELECTABLE_NOT_SELECTED,
    SELECTABLE_SELECTED,
    NOT_SELECTABLE_SELECTED,
    NOT_SELECTABLE_NOT_SELECTED
};


// Corresponding MHEG datatype: Echo-Mode
//=========================================================================
enum EchoMode {
    ITSELF,
    HIDDEN
};


// Interface: RtContentOrPresentableSocket Operation: setEntryFieldInteractionStyle
// Corresponding MHEG datatype: Echo-Style
//=========================================================================
enum EchoStyleTag { MODE_TAG, SPECIFIED_TAG };
union EchoStyle
switch (EchoStyleTag){
    case MODE_TAG:
        EchoMode
            mode;
    case SPECIFIED_TAG:
        string
            specified;
};


// Corresponding MHEG datatype: Channel-Reference
//=========================================================================
enum ChannelReferenceTag { CHANNEL_IDENTIFIER_TAG, ALIAS_TAG, NULL_CHANNEL_REFERENCE_TAG };
union ChannelReference
switch (ChannelReferenceTag){
    case CHANNEL_IDENTIFIER_TAG:
        ChannelIdentifier
            channel_identifier;
    case ALIAS_TAG:
        Alias
            alias;
};


// Corresponding MHEG datatype: Interval
//=========================================================================
struct Interval {
    sequence<long,1>
        start_point;
    sequence<long,1>
        end_point;
};


// Corresponding MHEG datatype: Generic-Volume-Range
//=========================================================================
struct GenericVolumeRange {
    sequence<long,1>
        maximum_volume;
    sequence<long,1>
        minimum_volume;
};


// Interface: Channel Operation: new
// Corresponding MHEG datatype: Original-Def-Declaration
//=========================================================================
struct OriginalDefDeclaration {
    sequence<long,1>
        generic_temporal_ratio;
    sequence<Interval,1>
```

```
        x_axis_interval;
    sequence<Interval,1>
        y_axis_interval;
    sequence<Interval,1>
        z_axis_interval;
    sequence<GenericVolumeRange,1>
        audible_volume_range_declaration;
};


// Interface: Channel Operation: getAvailability
// Corresponding MHEG datatype: Channel-Status-ValueCHANNEL-STATUS-VALUE-
//=====================================================================
enum ChannelStatusValue {
    CHANNEL_STATUS_VALUE_AVAILABLE,
    CHANNEL_STATUS_VALUE_NOT_AVAILABLE,
    CHANNEL_STATUS_VALUE_PROCESSING
};


// Interface: Channel Operation: setPerceptability
// Interface: Channel Operation: getPerceptability
// Corresponding MHEG datatype: Channel-Perceptability-Values
//=====================================================================
enum ChannelPerceptabilityValue {
    ON,
    OFF
};


// Interface: NotificationManager Operation: getNotification
// Interface: MhContent Operation: getData
// Corresponding MHEG datatype: Generic-Value
//=====================================================================
enum GenericValueTag { BOOLEAN_FIELD_TAG, NUMERIC_TAG, STRING_FIELD_TAG, GENERIC_LIST_TAG,
UNSPECIFIED_TAG };
union GenericValue
switch (GenericValueTag){
    case BOOLEAN_FIELD_TAG:
        boolean
            boolean_field;
    case NUMERIC_TAG:
        long
            numeric;
    case STRING_FIELD_TAG:
        string
            string_field;
    case GENERIC_LIST_TAG:
        sequence<GenericValue>
            generic_list;
};


// Corresponding MHEG datatype: Generic-String
//=====================================================================
enum GenericStringTag { GENERIC_STRING_CONSTANT_TAG, GENERIC_STRING_UNSPECIFIED_TAG };
union GenericString
switch (GenericStringTag){
    case GENERIC_STRING_CONSTANT_TAG:
        string
            constant;
};


// Interface: Socket Operation: setVisibleDurationPosition
// Corresponding MHEG datatype: Visible-Duration
//=====================================================================
enum VisibleDurationPositionTag { VISIBLE_DURATION_POSITION_SPECIFIED_TEMPORAL_POINT_TAG,
VISIBLE_DURATION_POSITION_LOGICAL_TEMPORAL_PD_POINT_TAG,
VISIBLE_DURATION_POSITION_LOGICAL_TEMPORAL_VD_POINT_TAG };
union VisibleDurationPosition
switch (VisibleDurationPositionTag){
    case VISIBLE_DURATION_POSITION_SPECIFIED_TEMPORAL_POINT_TAG:
        long
            specified_temporal_point;
    case VISIBLE_DURATION_POSITION_LOGICAL_TEMPORAL_PD_POINT_TAG:
        long
            logical_temporal_PD_point;
    case VISIBLE_DURATION_POSITION_LOGICAL_TEMPORAL_VD_POINT_TAG:
        long
            logical_temporal_VD_point;
};


// Interface: RtComponentOrSocket Operation: getRGS
```

```
// Corresponding MHEG datatype: none
//=========================================================================
enum RGSValueTag { RGS_VALUE_CHANNEL_IDENTIFIER_TAG, RGS_VALUE_NULL_CHANNEL_TAG,
RGS_VALUE_PRGS_TAG };
union RGSValue
switch (RGSValueTag){
    case RGS_VALUE_CHANNEL_IDENTIFIER_TAG:
        ChannelIdentifier
            channel_identifier;
};


// Corresponding MHEG datatype: Generic-Numeric
//=========================================================================
enum GenericNumericTag { GENERIC_NUMERIC_CONSTANT_TAG, GENERIC_NUMERIC_UNSPECIFIED_TAG };
union GenericNumeric
switch (GenericNumericTag){
    case GENERIC_NUMERIC_CONSTANT_TAG:
        long
            constant;
};


// Interface: RtComponentOrSocket Operation: getSelectionMode
// Corresponding MHEG datatype: none
//=========================================================================
enum SelectionModeValueTag { USER_INTERACTION_TAG, NO_SELECTION_TAG, MHEG_ACTION_TAG,
USING_APPLICATION_ACTION_TAG };
union SelectionModeValue
switch (SelectionModeValueTag){
    case USER_INTERACTION_TAG:
        unsigned long
            user_interaction;
};


// Interface: RtComponentOrSocket Operation: getModificationMode
// Corresponding MHEG datatype: none
//=========================================================================
enum ModificationModeValueTag { MODIFICATION_MODE_VALUE_USER_INTERACTION_TAG,
MODIFICATION_MODE_VALUE_NO_MODIFICATION_TAG, MODIFICATION_MODE_VALUE_MHEG_ACTION_TAG,
MODIFICATION_MODE_VALUE_USING_APPLICATION_ACTION_TAG, MODIFICATION_MODE_VALUE_CHILD_TAG };
union ModificationModeValue
switch (ModificationModeValueTag){
    case MODIFICATION_MODE_VALUE_USER_INTERACTION_TAG:
        unsigned long
            user_interaction;
};


// Corresponding MHEG datatype: External-Identifier
//=========================================================================
enum ExternalIdentifierTag { EXTERNAL_LONG_ID_TAG, PUBLIC_ID_TAG, SYSTEM_ID_TAG };
union ExternalIdentifier
switch (ExternalIdentifierTag){
    case EXTERNAL_LONG_ID_TAG:
        ExternalLongIdentifier
            external_long_id;
    case PUBLIC_ID_TAG:
        PublicIdentifier
            public_id;
    case SYSTEM_ID_TAG:
        SystemIdentifier
            system_id;
};

// Corresponding MHEG datatype: Container-Tail
//=========================================================================
struct ContainerTail {
    sequence<long>
        indexes;
    enum ContainerTailTag { INDEX_TAG, CONTAINER_CHILD_REF_TAG } tag;
    union ContainerTail
    switch (ContainerTailTag){
        case INDEX_TAG:
            long
                index;
        case CONTAINER_CHILD_REF_TAG:
            ContainerChildReference
                container_child_ref;
    } end;
};

// Corresponding MHEG datatype: Specified-Sizes
//=========================================================================
```

```
struct SpecifiedSizes {
    sequence<GenericNumeric,1>
        x_axis_length;
    sequence<GenericNumeric,1>
        y_axis_length;
    sequence<GenericNumeric,1>
        z_axis_length;
};


// Corresponding MHEG datatype: Attachment-Attribute
//=========================================================================
enum AttachmentAttributeTag { SPECIFIED_POSITION_TAG, LOGICAL_POSITION_TAG };
union AttachmentAttribute
switch (AttachmentAttributeTag){
    case SPECIFIED_POSITION_TAG:
        GenericNumeric
            specified_position;
    case LOGICAL_POSITION_TAG:
        GenericNumeric
            logical_position;
};



// Corresponding MHEG datatype: Length-Attribute
//=========================================================================
enum LengthAttributeTag { SPECIFIED_LENGTH_TAG, RELATIVE_LENGTH_TAG };
union LengthAttribute
switch (LengthAttributeTag){
    case SPECIFIED_LENGTH_TAG:
        GenericNumeric
            specified_length;
    case RELATIVE_LENGTH_TAG:
        GenericNumeric
            relative_length;
};


// Interface: RtComponentOrSocket Operation: getPS
// Interface: RtComponentOrSocket Operation: getPSAP
// Interface: RtComponentOrSocket Operation: getVS
// Interface: RtComponentOrSocket Operation: getVSIAP
// Interface: RtComponentOrSocket Operation: getVSIAPPosition
// Interface: RtComponentOrSocket Operation: getVSEAP
// Interface: RtComponentOrSocket Operation: getVSEAPPosition
// Corresponding MHEG datatype: Specified-Position
//=========================================================================
struct SpecifiedPosition {
    GenericNumeric
        x_point;
    GenericNumeric
        y_point;
    GenericNumeric
        z_point;
};


// Interface: RtComponentOrSocket Operation: setPresentationPriority
// Corresponding MHEG datatype: Presentation-Priority
//=========================================================================
enum PresentationPriorityTag { GENERIC_NUMERIC_TAG, PRIORITY_LEVEL_TAG };
union PresentationPriority
switch (PresentationPriorityTag){
    case GENERIC_NUMERIC_TAG:
        GenericNumeric
            generic_numeric;
    case PRIORITY_LEVEL_TAG:
        PriorityLevel
            priority_level;
};


// Interface: RtComponentOrSocket Operation: setTimestones
// Corresponding MHEG datatype: Timestone
//=========================================================================
struct Timestone {
    long
        timestone_identifier;
    TimestonePosition
        timestone_position;
};


// Interface: RtComponentOrSocket Operation: setVisibleSize
```

```
// Corresponding MHEG datatype: none
//=========================================================================
enum VSTag { X_SIZE_ATTRIBUTE_TAG, Y_SIZE_ATTRIBUTE_TAG, Z_SIZE_ATTRIBUTE_TAG };
union VS
switch (VSTag){
    case X_SIZE_ATTRIBUTE_TAG:
        SizeAttribute
            x_size_attribute;
    case Y_SIZE_ATTRIBUTE_TAG:
        SizeAttribute
            y_size_attribute;
    case Z_SIZE_ATTRIBUTE_TAG:
        SizeAttribute
            z_size_attribute;
};


// Interface: RtComponentOrSocket Operation: setAttachmentPoint
// Corresponding MHEG datatype: none
//=========================================================================
struct AttachmentPoint {
    sequence<AttachmentAttribute,1>
        x_attachment;
    sequence<AttachmentAttribute,1>
        y_attachment;
    sequence<AttachmentAttribute,1>
        z_attachment;
};


// Interface: RtComponentOrSocket Operation: setAttachmentPointPosition
// Corresponding MHEG datatype: Lengths
//=========================================================================
struct Lengths {
    sequence<LengthAttribute,1>
        x_length;
    sequence<LengthAttribute,1>
        y_length;
    sequence<LengthAttribute,1>
        z_length;
};


// Interface: RtMultiplexedContentOrPresentableSocket Operation: getStreamChosen
// Corresponding MHEG datatype: none
//=========================================================================
enum StreamValueTag { STREAM_IDENTIFIER_TAG, NO_STREAM_CHOSEN_TAG };
union StreamValue
switch (StreamValueTag){
    case STREAM_IDENTIFIER_TAG:
        StreamIdentifier
            stream_identifier;
};

// Interface: MhContent Operation: setData
// Corresponding MHEG datatype: Data-Element
//=========================================================================
struct DataElement {
    sequence<long>
        element_list_index;
    GenericValue
        generic_value;
};


// Interface: NotificationManager Operation: getNotification
// Interface: MhObject Operation: bind
// Interface: MhObject Operation: prepare
// Interface: MhGenericContent Operation: copy
// Corresponding MHEG datatype: Mh-Object-Reference
//=========================================================================
struct MhObjectReference {
enum MhObjectReferenceHeadTag { MHEG_IDENTIFIER_TAG, EXTERNAL_IDENTIFIER_TAG,
ALIAS_TAG, NULL_OBJECT_REF_TAG } head_tag;
union MhObjectReferenceHead
switch (MhObjectReferenceHeadTag){
    case MHEG_IDENTIFIER_TAG:
        MHEGIdentifier
            mheg_identifier;
    case EXTERNAL_IDENTIFIER_TAG:
        ExternalIdentifier
            external_identifier;
    case ALIAS_TAG:
        Alias
            alias;
```

```
} head;
enum MhObjectReferenceTailTag { CONTAINER_ELEMENT_REFERENCE_TAG, OTHER_REFERENCE_TAG
} tail_tag;
union MhObjectReferenceTail
switch (MhObjectReferenceTailTag){
    case CONTAINER_ELEMENT_REFERENCE_TAG:
        ContainerTail
            container_tail;
} tail;
};

// Interface: RtComponentOrSocket Operation: setPerceptibleSizeProjection
// Corresponding MHEG datatype: Perceptible-Size-Projection
//=========================================================================
struct PerceptibleSizeProjection {
    sequence<PerceptibleProjection,1>
        x_perceptible_size_projection;
    sequence<PerceptibleProjection,1>
        y_perceptible_size_projection;
    sequence<PerceptibleProjection,1>
        z_perceptible_size_projection;
};


// Corresponding MHEG datatype: Rt-Object-Number-Reference
//=========================================================================
enum RtObjectNumberReferenceTag { RT_OBJECT_NUMBER_TAG, RT_DYNAMIC_REFERENCE_TAG };
union RtObjectNumberReference
switch (RtObjectNumberReferenceTag){
    case RT_OBJECT_NUMBER_TAG:
        long
            rt_object_number;
    case RT_DYNAMIC_REFERENCE_TAG:
        RtDynamicReference
            rt_dynamic_reference;
};


// Interface: RtObject Operation: bind
// Interface: RtObject Operation: new
// Corresponding MHEG datatype: Rt-Object-Reference
//=========================================================================
struct RtObjectReference {
    MhObjectReference
        model_object_reference;
    RtObjectNumberReference
        rt_object_number_reference;
};


// Corresponding MHEG datatype: Rt-Reference
//=========================================================================
enum RtReferenceTag { RT_REFERENCE_RT_OBJECT_REFERENCE_TAG, RT_REFERENCE_ALIAS_TAG,
RT_REFERENCE_NULL_RT_OBJECT_TAG };
union RtReference
switch (RtReferenceTag){
    case RT_REFERENCE_RT_OBJECT_REFERENCE_TAG:
        RtObjectReference
            rt_object_reference;
    case RT_REFERENCE_ALIAS_TAG:
        Alias
            alias;
};


// Corresponding MHEG datatype: Socket-Tail
//=========================================================================
struct SocketTail {
    sequence<long>
        indexes;
    enum SocketTailTag { INDEX_TAG, SUB_SOCKET_REF_TAG } tag;
    union SocketTail
    switch (SocketTailTag){
        case INDEX_TAG:
            long
                index;
        case SUB_SOCKET_REF_TAG:
            SubSocketReference
                sub_socket_ref;
    } end;
};


// Corresponding MHEG datatype: Indexed-Child-Socket
//=========================================================================
```

```
struct IndexedChildSocket {
    long
        index;
    SocketTail
        tail;
};


// Interface: Socket Operation: bind
// Interface: Socket Operation: getIdentification
// Corresponding MHEG datatype: Socket-Identification
//=====================================================================
struct SocketIdentification {
    RtReference
        rt_composite_reference;
    SocketTail
        socket_tail;
};


// Interface: Socket Operation: bind
// Corresponding MHEG datatype: Socket-Reference
//=====================================================================
enum SocketReferenceTag { SOCKET_REFERENCE_SOCKET_IDENT_TAG, SOCKET_REFERENCE_ALIAS_TAG };
union SocketReference
switch (SocketReferenceTag){
    case SOCKET_REFERENCE_SOCKET_IDENT_TAG:
        SocketIdentification
            socket_ident;
    case SOCKET_REFERENCE_ALIAS_TAG:
        Alias
            alias;
};


// Corresponding MHEG datatype: Rt-Object-Socket-Reference
//=====================================================================
enum RtObjectSocketReferenceTag { RT_REFERENCE_TAG, SOCKET_REFERENCE_TAG };
union RtObjectSocketReference
switch (RtObjectSocketReferenceTag){
    case RT_REFERENCE_TAG:
        RtReference
            rt_reference;
    case SOCKET_REFERENCE_TAG:
        SocketReference
            socket_reference;
};


// Interface: RtCompositeOrStructuralSocket Operation: setScrollingListInteractionStyle
// Interface: RtContentOrPresentableSocket Operation: setSliderInteractionStyle
// Interface: RtContentOrPresentableSocket Operation: setEntryFieldInteractionStyle
// Interface: Channel Operation: getAssignedPerceptibles
// Corresponding MHEG datatype: Perceptible-Reference
//=====================================================================
enum PerceptibleReferenceTag { RT_COMPONENT_REFERENCE_TAG, RT_SOCKET_REFERENCE_TAG };
union PerceptibleReference
switch (PerceptibleReferenceTag){
    case RT_COMPONENT_REFERENCE_TAG:
        RtReference
            rt_component_reference;
    case RT_SOCKET_REFERENCE_TAG:
        SocketReference
            rt_socket_reference;
};


// Interface: RtCompositeOrStructuralSocket Operation: setScrollingListInteractionStyle
// Corresponding MHEG datatype: Separator
//=====================================================================
enum SeparatorTag { NO_TAG, YES_DEFAULT_TAG, SEPARATOR_PIECE_TAG };
union Separator
switch (SeparatorTag){
    case SEPARATOR_PIECE_TAG:
        PerceptibleReference
            separator_piece;
};


// Interface: RtCompositeOrStructuralSocket Operation: setMenuInteractionStyle
// Corresponding MHEG datatype: Association
//=====================================================================
struct Association {
    sequence<SocketReference,1>
        title;
```

```
    sequence<Separator,1>
        separator;
    sequence<SocketReference,1>
        submenu;
    sequence<PresentationPersistence,1>
        submenu_presentation_persistence;
    sequence<Orientation,1>
        submenu_orientation;
};


// Interface: RtSocket Operation: plug
// Corresponding MHEG datatype: Plug-In
//=========================================================================
enum PlugInTag { PLUG_IN_RT_COMPONENT_REFERENCE_TAG, PLUG_IN_COMPONENT_REFERENCE_TAG,
PLUG_IN_LABEL_TAG };
union PlugIn
switch (PlugInTag){
    case PLUG_IN_RT_COMPONENT_REFERENCE_TAG:
        RtObjectReference
            rt_component_reference;
    case PLUG_IN_COMPONENT_REFERENCE_TAG:
        MhObjectReference
            component_reference;
    case PLUG_IN_LABEL_TAG:
        GenericString
            label;
};


// Interface: RtComponentOrSocket Operation: getVSEAPPosition
// Corresponding MHEG datatype: none
//=========================================================================
enum ReferencePointTag { VSEAP_POSITION_ORIGIN_RGS_TAG, VSEAP_POSITION_ORIGIN_CGS_TAG,
VSEAP_POSITION_SAME_RGS_COMPONENT_TAG, VSEAP_POSITION_SAME_CGS_COMPONENT_TAG,
VSEAP_POSITION_SPECIFIED_POSITION_TAG };
union ReferencePoint
switch (ReferencePointTag){
    case VSEAP_POSITION_SAME_RGS_COMPONENT_TAG:
        RtObjectSocketReference
            same_RGS_component;
    case VSEAP_POSITION_SAME_CGS_COMPONENT_TAG:
        RtObjectSocketReference
            same_CGS_component;
    case VSEAP_POSITION_SPECIFIED_POSITION_TAG:
        SpecifiedPosition
            specified_position;
};


// Interface: RtScript Operation: setParameters
// Corresponding MHEG datatype: Parameter
//=========================================================================
enum ParameterTag { GENERIC_VALUE_TAG, MH_OBJECT_REFERENCE_TAG };
union Parameter
switch (ParameterTag){
    case GENERIC_VALUE_TAG:
        GenericValue
            generic_value;
    case MH_OBJECT_REFERENCE_TAG:
        MhObjectReference
            mh_object_reference;
};


// Interface: RtObjectOrSocket Operation: setGlobalBehaviour
// Corresponding MHEG datatype: Global-Behaviour
//=========================================================================
enum GlobalBehaviourTag { GLOBAL_BEHAVIOUR_RT_REFERENCE_TAG, GLOBAL_BEHAVIOUR_GENERIC_LIST_TAG,
GLOBAL_BEHAVIOUR_UNSPECIFIED_TAG };
union GlobalBehaviour
switch (GlobalBehaviourTag){
    case GLOBAL_BEHAVIOUR_RT_REFERENCE_TAG:
        RtReference
            rt_reference;
    case GLOBAL_BEHAVIOUR_GENERIC_LIST_TAG:
        GenericValue
            generic_list;
};


// Interface: RtComponentOrSocket Operation: setVisibleSizesAdjustment
// Corresponding MHEG datatype: Adjustment-Policy
//=========================================================================
enum AdjustmentPolicyTag { ADJUSTMENT_POLICY_COMPONENT_REFERENCE_TAG,
```

```
ADJUSTMENT_POLICY_SPECIFIED_TAG, ADJUSTMENT_POLICY_GREATEST_TAG, ADJUSTMENT_POLICY_SMALLEST_TAG
};
union AdjustmentPolicy
switch (AdjustmentPolicyTag){
    case ADJUSTMENT_POLICY_COMPONENT_REFERENCE_TAG:
        RtObjectSocketReference
            component_reference;
    case ADJUSTMENT_POLICY_SPECIFIED_TAG:
        SpecifiedSizes
            specified;
};


// Interface: RtObject Operation: bind
// Interface: RtObject Operation: new
// Interface: RtObject Operation: getIdentifier
// Corresponding MHEG datatype: none
//=====================================================================
struct RtObjectIdentifier {
    MHEGIdentifier
        model_object_id;
    long
        rt_object_number;
};


// Interface: RtGenericContentOrPresentableSocket Operation: setButtonInteractionStyle
// Corresponding MHEG datatype: Alternate-Presentation-State
//=====================================================================
struct AlternatePresentation {
    PresentationState
        presentation_state;
    PerceptibleReference
        perceptible_target;
};
```

7.1.37 Exceptions

7.1.37.1 `InvalidTarget` exception

Description:

The `InvalidTarget` exception is raised when the targeted MHEG entity is not available. The `period` member returns the current period of the target.

7.1.37.2 `InvalidParameter` exception

Description:

The `InvalidParameter` exception is raised when the value of one of the parameters prohibits the normal execution of the action. The `completion_status` member indicates whether the action was completed (with a default value assigned to the inadequate parameter) or not. The `parameter_number` member identifies the rank of the invalid parameter.

7.1.37.3 `NotBound` exception

Description:

The `NotBound` exception is raised when the interface object instance is not bound with an MHEG entity.

7.1.37.4 `AlreadyBound` exception

Description:

The `AlreadyBound` exception is raised when the interface object instance is already bound with an MHEG entity. The `entity_identifier` member identifies the bound entity.

7.1.37.5 IDL definition

```
exception InvalidTarget {
    unsigned short period;
};

enum CompletionStatus { YES, NO};

exception InvalidParameter {
    CompletionStatus completion_status;
    unsigned short period;
};

typedef long EntityIdentifier
exception AlreadyBound {
    EntityIdentifier entity_identifier;
};

exception NotBound {};
```

7.2 Optional primitives

The following objects shall be used to modify MHEG objects in form b:

- mhAction
- mhComposite
- mhContainer
- mhContent
- mhDescriptor
- mhLink
- mhMultiplexedContent
- mhScript

Annex A (normative): Complete IDL definition of the MHEG API

The complete IDL definition of the MHEG API is contained in an archive file named 714__e1.lzh which can be found on the electronic version which accompanies this ETS.

History

Document history			
April 1996	Public Enquiry	PE 105:	1996-04-08 to 1996-08-30
February 1997	Vote	V 9715:	1997-02-11 to 1997-04-11
May 1997	First Edition		

ISBN 2-7437-1426-3
Dépôt légal : Mai 1997