EUROPEAN TELECOMMUNICATION STANDARD

DRAFT
pr ETS 300 715

May 1996

First Edition

Source: ETSI TC-TE

Reference: DE/TE-01047

ICS: 33.020

Key words: MHEG, multimedia, hypermedia, script

Terminal Equipment (TE); Multimedia and Hypermedia Experts Group (MHEG); Script Interchange Representation (SIR)

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

*

Contents

Foreword

This draft European Telecommunication Standard (ETS) has been produced by the Terminal Equipment (TE) Technical Committee of the European Telecommunications Standards Institute (ETSI), and is now submitted for the Public Enquiry phase of the ETSI standards approval procedure.

This draft ETS was presented to ISO SC29 WG12 and is now under CD-Ballot procedure within ISO.

The title of this document in ISO is:

TITLE: Combined CD Registration and CD Consideration Ballot on ISO/IEC CD 13522-3, Information technology — Coding of multimedia and hypermedia information — Part 3: — MHEG extensions for scripting language support, — Extensions for script object interchange

This draft ETS was developed by ETSI PT63 jointly with ISO SC29 WG12.

Proposed transposition dates	
Date of latest announcement of this ETS (doa):	3 months after ETSI publication
Date of latest publication of new National Standard or endorsement of this ETS (dop/e):	6 months after doa
Date of withdrawal of any conflicting National Standard (dow):	6 months after doa

Introduction

Multimedia and Hypermedia information coding Experts Group (MHEG) part 1 (ISO/IEC 13522-1 [1]) is a generic International Standard/Recommendation, which specifies the coded representation of multimedia/hypermedia information objects (MHEG objects) for interchange as final form units within or across services and applications, by any means of interchange including local area networks, wide area telecommunication or broadcast networks, storage media, etc.

MHEG objects will usually be produced by computer tools taking multimedia applications designed using multimedia scripting languages as a source form. In this context, one of the MHEG object classes, the script class, is intended to complement the other MHEG classes in expressing the functionality commonly supported by scripting languages. Script objects allow the expression of more powerful control mechanisms and the description of more complex relationships between MHEG objects than can be expressed by MHEG action and link objects alone. Furthermore, script objects allow the expression of access and interaction with external services provided by the run-time environment.

MHEG part 1 defines the coded representation for script objects in an open manner so as not to exclude the transport of proprietary script code. Script objects encapsulate scripts that may be encoded in any encoding format as registered according to MHEG part 4 (ISO/IEC 13522-4) [6].

Blank page

1 Scope

This draft European Telecommunication Standard (ETS) is intended to extend the coded representation of the Multimedia and Hypermedia information coding Experts Group (MHEG) script object class.

The draft ETS specifies the MHEG Script Interchange Representation (MHEG-SIR) for the contents of script objects.

The MHEG-SIR consists of the encoding of the script data component of the MHEG script class. It applies to MHEG part 1 (ISO/IEC 13522-1 [1]) conforming script objects whose script classification component is script and whose script encoding identification component is MHEG-SIR.

NOTE: These values will be registered according to MHEG part 4 (ISO/IEC 13522-4 [6]) provisions.

MHEG engines are system or application components which handle, interpret and present MHEG objects. This ETS also specifies the semantics of MHEG-SIR interchanged scripts. The semantics are defined in terms of minimum requirements on the behaviour of MHEG engines which support the interpretation of MHEG-SIR scripts.

2 Normative references

This ETS incorporates by dated or undated reference, provisions from other publications. These 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 application referred to applies.

[1] ISO/IEC 13522-1/ITU-T Recommendation T.171: "Information technology - Coding of Multimedia and Hypermedia Information: - Part 1: MHEG object representation - Base Notation".

[2] ISO/IEC 8824-1 (1995)/ITU-T Recommendation X.680 (1995): "Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation".

[3] ISO/IEC 8825-1 (1994)/ITU-T Recommendation X.690 (1995): "Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)".

[4] ITU-T Draft Recommendation T.177: "Terminal Equipment (TE); MHEG Application Programming Interface (API)".

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

[6] ISO/IEC 13522-4: "Information technology - Coding of Multimedia and Hypermedia Information: - Part 4: Registration procedure for MHEG format identifiers".

[7] ISO/IEC 10646-1 (1993): "Information technology - Universal Multiple-Octet Coded Character Set (UCS) - Part1: Architecture and Basic Multilingual Plane".

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

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

[10] Draft ETS 300 714: "Terminal Equipment (TE); Application Programming manupulation of Multimedia and Hypermedia information objects".

3 Definitions and abbreviations

3.1 Definitions

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

NOTE: The definitions given in ISO/IEC 8824-1 [2], ISO/IEC 8825-1 [3] are also applicable to this ETS.

attribute: a) An MHEG attribute;
b) An IDL attribute (q.v.).

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.

calling stack: The MHEG-SIR calling stack (q.v.).

constant: An MHEG-SIR constant (q.v.).

constructed type: An MHEG-SIR constructed type (q.v.).

data identifier: An MHEG-SIR data identifier (q.v.).

exception: An MHEG-SIR exception (q.v.).

frame: A set of elements on the calling stack that define the current execution context. Multiple frames may appear on the stack.

function: An MHEG-SIR function (q.v.).

function identifier: An MHEG-SIR function identifier (q.v.).

global variable: An MHEG-SIR global variable (q.v.).

handler: An MHEG-SIR handler (q.v.).

Hypermedia (ADJ.): Featuring access to monomedia and multimedia information by interaction with explicit links.

instruction: An MHEG-SIR instruction (q.v.).

instruction execution unit: An MHEG-SIR instruction execution unit (q.v.).

interchanged script: The coded representation of the script data attribute of an MHEG script object.

Interface Definition Language (IDL): A formal notation that is used to specify types and objects through the definition of the interface that they provide. Defined by the OMG, IDL is under currently under an international standardisation process (see ISO/IEC 14750-1 [8]).

IDL attribute: A named and typed association between an object and a value. An IDL attribute A is made visible to clients as a pair of operations: An accessor (get) and a modifier (set). Read-only attributes only provide an accessor.

IDL exception: the definition, using IDL, of a client operation in an invoking script object that is called when an error occurs during the performance of the request to an IDL operation. IDL exceptions are defined in IDL modules and may have members (parameters).

IDL instance: An instance of an IDL interface is an object which provides the operations, signatures and semantics specified by that interface. The creation and management of instances is implementation specific.

IDL interface: The definition, using the IDL, of an object or non-object type as a set of operations and attributes.

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 IDL object may be implemented as a computational entity that encapsulates state and operations (internally implemented as data and processing instructions) and responds to requester services.

IDL operation: A service that can be requested. An IDL operation is defined by a name, a signature, which defines the type of its parameters and return value, and the list of exceptions that its invocation may raise.

local variable: An MHEG-SIR local variable (q.v.).

message: An MHEG-SIR message (q.v.).

mh-script: An internal representation of an available MHEG script object within an MHEG engine.

MHEG action: An operation applying to MHEG objects.

MHEG action object: An MHEG object that describes MHEG actions.

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

MHEG conforming object: An information object whose coded representation conforms to the provisions of ISO/IEC 13522-1 [1].

MHEG elementary action: One of the basic operations defined by ISO/IEC 13522-1 [1]; MHEG elementary actions map MHEG-API primitives.

MHEG engine: A process or a set of processes able to interpret MHEG objects which conform to ISO/IEC 13522-1 [1].

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

MHEG link: An MHEG object which defines spatio-temporal relationships between MHEG objects expressed in terms of trigger conditions and actions.

MHEG object: A coded representation of an instance of an MHEG object class, as defined by ISO/IEC 13522-1 [1].

MHEG script class: An MHEG class defining a structure to interchange script data in a specified encoded form.

MHEG script object: The coded representation of an instance of an MHEG script class, as defined by ISO/IEC 13522-1 [1].

MHEG-1:
a) Part 1 of ISO/IEC 13522-1, i.e. ISO/IEC 13522-1 [1];
b) the coded representation defined by ISO/IEC 13522-1 [1].

MHEG-4: a) Part 4 of ISO/IEC 13522-1, i.e. ISO/IEC 13522-4 [6];
b) the registration mechanisms defined by ISO/IEC 13522-4 [6].

MHEG-API: The API provided by an MHEG engine to MHEG applications for the manipulation of MHEG objects, as defined by ITU-T Recommendation T.177 [4].

MHEG-S (ADJ.): Applies to entities that conform to the provisions of this ETS.

MHEG-S application: An MHEG application which interchanges scripts within itself and/or with other applications as the script data component of MHEG script objects according to the representation specified by this ETS.

MHEG-S conforming object: An information object whose coded representation conforms to the provisions of this ETS.

MHEG-S conforming implementation: An MHEG-S engine whose implementation conforms to the provisions of this ETS.

MHEG-S conforming interchanged script: An interchanged script which conforms to the provisions of this ETS.

MHEG-S engine: An MHEG engine which processes and interprets MHEG-S interchanged scripts.

MHEG-S profile: A profile of this ETS.

MHEG-SIR: a) The standard Script Interchange Representation defined by this ETS;
b) (adj.) applies to an entity defined as part of the MHEG-SIR.

MHEG-SIR calling stack: A stack which contains a call frame for each active function invocation.

MHEG-SIR code: An encoded sequence of MHEG-SIR instructions.

MHEG-SIR constant: A static, typed, and named value declared within an interchanged script, whose value is globally accessible and unchanged throughout the execution of the script.

MHEG-SIR constructed type: A type whose description uses one of the following type constructors: sequence, array, union, enumerated or structure.

MHEG-SIR data identifier: A value which unambiguously identifies the name of data element of an interchanged script (constant, global variable or local variable).

MHEG-SIR exception: A message triggered during the invocation of a service.

MHEG-SIR function: A named code sequence whose execution can be invoked by an interchanged script. MHEG-SIR functions are routines, predefined functions and services.

MHEG-SIR function identifier: An integer which uniquely identifies a function within an interchanged script.

MHEG-SIR global variable: A variable with global scope.

MHEG-SIR instruction: An elementary unit of script code that consists of an op-code followed by zero or more operands.

MHEG-SIR instruction execution unit: A virtual component of a script interpreter responsible for the execution of MHEG-SIR code.

MHEG-SIR local variable: A variable with local scope.

MHEG-SIR message: An event, either predefined or declared within an interchanged script, which may be received by the script interpreter during the execution of the script. Messages include MHEG actions and exceptions.

MHEG-SIR object reference: An opaque handle to an IDL object which is dereferenced in order to manipulate the object.

MHEG-SIR operand: A parameter of an instruction, whose encoding follows the op-code of the instruction.

MHEG-SIR package: A set of external functions, provided by a module of the run-time environment, accessible to an rt-script and declared within an interchanged script. A package consists of services and exceptions.

MHEG-SIR parameter: A piece of data exchanged with a function call, a message or an instruction.

MHEG-SIR parameter stack: A dynamic memory area of the MHEG-SIR virtual machine used to provide parameters to and retrieve results of instructions.

MHEG-SIR predefined type: A type whose description and identifier are predefined by this ETS, and thus need not be declared within interchanged scripts.

MHEG-SIR primitive type: A basic type, whose description and identifier are predefined by this ETS, and thus need not be declared within interchanged scripts.

MHEG-SIR routine: A function which is declared within an interchanged script together with the pseudo-code that defines its semantics.

MHEG-SIR script code execution unit: A virtual component of an MHEG-SIR script interpreter that deals with the interpretation of MHEG-SIR code.

MHEG-SIR script interpreter: A part of an MHEG engine which deals with the handling and interpretation of interchanged scripts.

MHEG-SIR service: An external function, declared within an interchanged script, whose implementation is made accessible to an rt-script by the run-time environment on the execution platform.

MHEG-SIR variable: A named and typed memory unit whose value can be changed at any time when its scope is active, and whose most recent value can be read.

MHEG-SIR virtual machine: An abstract description of the instruction execution engine of an MHEG-SIR script interpreter in interpreting an MHEG-SIR interchanged script.

Multimedia (ADJ.): That which handles several types of representation media.

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

object reference: An MHEG-SIR object reference (q.v.).

operand: An MHEG-SIR operand (q.v.).

package: An MHEG-SIR package (q.v.).

parameter: An MHEG-SIR parameter (q.v.).

parameter stack: the MHEG-SIR parameter stack (q.v.).

platform mapping specification: A specification of how MHEG-S engine implementations shall map IDL specifications to run-time environment components on one type of platform.

predefined type: An MHEG-SIR predefined type (q.v.).

primitive type: An MHEG-SIR primitive type (q.v.).

pseudo-code: Interpreted code that resembles executable code.

queue: A collection of elements which are inserted and removed in First-In-First-Out (FIFO) order.

routine: An MHEG-SIR routine.

rt-script: A run-time instance (or copy) of a model mh-object, created and handled by an MHEG engine in the purpose of presentation.

scope: The context of reference for an object or variable. An object with global scope can be referenced by any script operation. An object with local scope can only be referenced in the local execution context or its descendants.

scripting language: A programming language intended for easy and rapid design of applications by non-professional programmers.

Script Interchange Representation (SIR): A coded representation used by an application to interchange scripts for the purpose of implementing dynamic behavior.

script code execution unit: MHEG-SIR script code execution unit (q.v.).

script interpreter: MHEG-SIR script interpreter (q.v.).

service: An MHEG-SIR service (q.v.).

stack: A collection of elements that are inserted and removed in Last-In-First-Out (LIFO) order.

variable: An MHEG-SIR variable (q.v.).

3.2 Abbreviations

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

API	Application Programming Interface
ASN.1	Abstract Syntax Notation One
BER	Basic Encoding Rules
CS	Calling Stack
CT	Constant Table
DER	Distinguished Encoding Rules
DID	Data IDentifier
DT	Data Table
EBNF	Extended Backus Naur Form
ER	Error Register
FID	Function IDentifier
FIFO	First In First Out
FP	Function Pointer
GT	Global variable Table
HT	Handler Definition Table
IDL	Interface Definition Language
IEC	International Electrotechnical Commission
IP	Instruction Pointer
IR	Instruction Register
LIFO	Last In First Out
LT	Local variable Table

MHEG	Multimedia and Hypermedia information coding Experts Group
MID	Message IDentifier
MPEG/DSM-CC	Moving Picture Experts Group - Digital Storage Media Command and Control
MQ	Message Queue
msb	Most Significant Bit
OMG	Object Management Group
PID	Package IDentifier
PS	Parameter Stack
PT	Package definition Table
QP	Queue Pointer
RT	Routine definition Table
rt	Run-time
SIR	Script Interchange Representation
SP	Stack Pointer
ST	Service definition Table
TE	Terminal Equipment
TID	Type IDentifier
TT	Type definition Table
TTCN	Tree And Tabular Combined Notation
TVL	Type, Length and Value
VD	Visible Duration
VM	Virtual Machine
VS	Visible Size
VT	Variable Table
XT	eXception definition Table

4 Conformance requirements

This ETS defines conformance requirements:

- on information objects;
- on MHEG engine implementations.

4.1 Information object conformance

An script object conforming to this ETS (called MHEG-S script object) shall meet the following criteria:

- its coded representation shall conform to the provisions of ISO/IEC 13522-1/ITU-T Recommendation T.171 [1];
- its coded representation shall encapsulate an interchanged script that conforms to the provisions of this ETS.

The information object conformance is evaluated on the information objects that are interchanged in the purpose of their execution on a terminal.

4.1.1 Profiles

This ETS defines no profiles.

NOTE: Profiles of this ETS may however be defined by other standards. In accordance with the profile definition framework, standardised profiles of this ETS should be at least as constraining - information objects claiming conformance to such profiles should at least conform to this ETS.

4.1.2 Encoding

Interchanged scripts shall be encoded according to the encoding rules defined by annex B.

4.1.3 Syntax

Interchanged scripts shall conform to the syntax defined by annex A.

4.1.4 Semantics

This ETS defines the semantics of interchanged scripts. This implies conformance requirements not on information objects, but on the behaviour of MHEG engines which will interpret interchanged scripts.

NOTE: This means that although a conforming script might not realise the semantics implied by its designer, the way conforming engines will behave in interpreting this script is predictable.

4.2 Implementation conformance

An implementation of this ETS (called MHEG-S implementation) is an MHEG engine implementation which supports the interpretation of interchanged script objects which conform to the provisions of this ETS.

There is no consideration of conformance for a system, an engine or a process as far as it is not related to the interpretation of interchanged scripts.

4.2.1 Conformance requirements

An MHEG-S Conforming Implementation shall meet all of the following criteria:

- it shall support all required behaviour defined in this ETS;

- it shall support all required interfaces defined in the platform mapping specification. Those interfaces shall support the behaviour described in this ETS and in the platform mapping specification;

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

4.2.2 Conformance documentation

A conformance document with the following information shall be available for an implementation claiming conformance to this ETS. The conformance document shall meet all of the following criteria:

- it shall list all the mandatory features required by this ETS, with reference to the appropriate clauses and subclauses;

- it shall either include the platform mapping specification to which the implementation conforms, or reference a registered platform mapping specification in an unambiguous way;

- it shall contain a statement that indicates the full names, numbers, and dates of the standards that apply;

- it shall state which of the optional features defined in this ETS and in the platform mapping specification are supported by the implementation.

- it shall describe the behaviour of the implementation for all implementation-defined features defined in this ETS and in the platform mapping 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 platform mapping 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 platform mapping specification shall be present in the conformance document.

4.3 Application conformance

An application of this ETS (called MHEG-S application) is an MHEG application which interchanges scripts within itself and/or with other applications as the script data component of MHEG script objects according to the encoded representation specified by this ETS.

All applications claiming conformance to this ETS shall fall within one of the categories defined in the following subclauses.

4.3.1 Strictly Conforming Application

An MHEG-S Strictly Conforming Application is an application that requires only the mandatory facilities described in this ETS.

4.3.2 Conforming Application

An MHEG-S Conforming Application is an application that differs from a Strictly Conforming Application in that it may use optional facilities described in this ETS or in the platform mapping specification, as well as non-standard facilities that are consistent with this ETS and with the platform mapping specification. Such an application shall fully document its requirements for these optional and extended facilities in addition to the documentation required of an MHEG-S Conforming Application.

4.4 Test Methods

Any measurement of conformance to this ETS shall be performed using test methods that conform to ISO/IEC 9646 [5] and to any additional requirements that may be imposed by the platform mapping specification.

5 Overview

This ETS extends the provisions of MHEG-1 so that MHEG objects and applications support the functionality of multimedia scripting languages in a standard manner. Considering the functionality that is already supported by MHEG-1, these extensions can be divided in two main topics, as described in subclause 7.2:

- data processing operations;
- access to external data and functions.

For the support of both topics, this ETS specifies:

- complete and detailed provisions for the encoding of interchanged scripts;
- the required behaviour of a script interpreter.

5.1 Description methodology

For the description of these provisions, this ETS follows the methodology used by MHEG-1 in considering four description levels:

- level a): informal text description;
- level b): precise description of semantics;
- level c): formal description of syntax;
- level d): formal description of encoding.

These levels are used in the following clauses as follows:

- level a): clauses 8 to 12;
- level b): clauses 13 to 15;

- level c): annex A;

- level d): annexes B to D.

NOTE: Informative annexes E, F and H also use level c) description.

5.2 Data processing operations

To deal with data processing operations, the MHEG-Script Interchange Representation (SIR) defines the structure of interchanged scripts that consist of data declarations and function declarations, the latter encapsulating sequences of instructions.

Clause 9 defines the elements of the MHEG-SIR virtual machine code.

Clause 10 defines the MHEG-SIR virtual machine, i.e. a model of how a script interpreter should perform interpretation of MHEG-SIR script code. This virtual machine is used afterwards to describe the semantics of MHEG-SIR instructions. Clause 10 states requirements on the functionality that script interpreters shall provide. However, there is no requirement on script interpreters on how they may implement this functionality.

Clause 13 describes the general structure and semantics of declarations, i.e. the way they are intended to be interpreted by script interpreters. These semantics are described using the virtual machine formalism introduced in clause 10.

Clause 14 describes the semantics of the MHEG-SIR instructions, i.e. the way they are intended to be interpreted by script interpreters. These semantics are described using the virtual machine formalism introduced in clause 10.

Annex A describes the precise syntax of interchanged scripts using ASN.1 notation.

Annex B formally describes the encoding of interchanged scripts.

Annex C lists the predefined elements of the MHEG-SIR and describes their encoding.

5.3 Access to external data and functions

To deal with access to external data and functions, the MHEG-SIR unifies the way external data and functions are viewed by script interpreters by use of the Interface Definition Language (IDL), which is used to describe interfaces in an abstract, language-independent way.

IDL clearly separates the way use of external data or functions is expressed by interchanged scripts (which is MHEG-SIR specific) from the way these data or functions are provided by the external environment (which is at least platform-dependent, and may be application-dependent). The MHEG-SIR thus defines how the interfaces are used, while the application is responsible for defining how they are provided.

To allow the script interpreter to co-operate with them, MHEG engines are assumed to provide access to the MHEG objects (data) and invocation of the MHEG actions (functions) through the MHEG-Application Programming Interface (API) defined (using the IDL) in ITU-T Recommendation T.177 [4]. The MHEG types and actions are predefined by the MHEG-SIR to allow compact coding and efficient interpretation of MHEG object manipulation.

To allow the script interpreter to co-operate with it, the run-time environment is assumed to provide access to its data and functions in a platform-dependent way which conforms to a platform mapping specification of IDL. This specification describes how IDL operations shall be provided on a particular platform for MHEG-S engines to use them as external services.

NOTE: Packages may be provided in the form of libraries, device drivers, operating system components, processes, telecommunication services, etc.

Clause 8 describes assumptions on the structure of MHEG-S engines and their relationships with their environment.

Clause 11 describes the general mechanisms that allow for access to external data and functions provided by the run-time environment.

Clause 12 describes the general mechanisms that allow for MHEG object manipulation.

Clause 15 describes the IDL mapping for MHEG-SIR, i.e. the mechanisms used by the MHEG-SIR representation to describe IDL packages and invoke IDL operations.

Annex D describes the IDL platform mapping specification form, i.e. the template for the document that needs to be filled in and registered to specify the platform-specific provisions that services provided by the run-time environment on this platform shall fulfil, and to which MHEG-S engines should conform if they wish to co-operate with services provided by the run-time environment on this platform.

6 MHEG/MHEG-S relationship

This clause introduces general assumptions about MHEG engines, which are used afterwards to describe the performance of a script interpreter and its relationships with its external environment.

MHEG-S engines shall provide the functionality described thereafter, in order to behave as expected in so far as interpretation of interchanged scripts is concerned.

However, there is no requirement on MHEG-S engines to implement this functionality as described.

NOTE: For instance, the MHEG engine functional components described thereafter need not correspond to actual (e.g. software) components of MHEG engine implementations.

6.1 Data entities

MHEG engines are assumed to handle MHEG entities: MHEG objects, mh-objects, run time (rt)-objects, interchanged MHEG objects, as described by ITU-T Recommendation T.177 [4].

NOTE: Descriptions of these entities are reproduced in annex G.

6.2 Functional entities

MHEG engines may be viewed as consisting of the following functional components:

- MHEG object parser: parses interchanged MHEG objects and transforms them into mh-objects under control of the mh-object manager;
- mh-object manager: controls the life cycle and allows access to all mh-objects;
- rt-object manager: controls the life cycle and allows access to all rt-objects;
- reference resolver: transforms an MHEG reference into a usable identifier or handle;
- link handler: watches active links and triggers the corresponding actions when their conditions become true;
- action interpreter: interprets MHEG elementary actions;
- script interpreter: parses MHEG-SIR interchanged scripts and interprets rt-scripts, provides access to the run-time environment;
- presentation agent: interface with the presentation environment, orders presentation of rt-contents, receives user selections and modifications;
- access agent: interface with the communication environment; provides access to interchanged MHEG objects and to content data.

6.3 MHEG-SIR script interpreter

Within an MHEG engine, the script interpreter is responsible for the following:

- parsing interchanged scripts (provided by the MHEG object parser);
- preparing the appropriate data structures for further execution of rt-scripts;
- executing script code;
- realising the default effect of MHEG actions targeted at mh- or rt-scripts;
- invoking the appropriate handler (in the script program) for these MHEG actions;
- forwarding MHEG elementary actions invoked by the script program to the action interpreter;
- managing interchanges with the run-time environment (locating and loading packages, invoking services, receiving messages, passing data) using the appropriate platform-specific communication mechanisms.

7 Main features of the MHEG-SIR

This ETS has been developed in order to respond to a number of requirements and constraints regarding

- the applications that may use it,
- the functionality it provides,
- its context of use by applications,
- its performance.

The MHEG-SIR specified by this ETS is endowed with the features described in the following subclauses.

7.1 Features of applications using MHEG-SIR

The MHEG-SIR fits the requirements of applications which feature:

- manipulation of MHEG multimedia presentation objects;
- external device control;
- data acquisition;
- access to external data;
- access to run-time services;
- computations, variable handling and control structures.

7.1.1 Manipulation of MHEG multimedia presentation objects

This feature is achieved through provision of the mechanisms described in subclause 7.2.2.

7.1.2 External device control

This feature is achieved through provision of the mechanisms described in subclause 7.2.2.

NOTE: In this context, a device driver is one service package that may be provided by the run-time environment.

7.1.3 External device control for data acquisition

This feature is achieved through provision of the mechanisms described in subclause 7.2.2.

NOTE: In this context, acquired data can be retrieved by an rt-script via asynchronous reaction to notification messages sent by the device driver.

7.1.4 Access to external data

This feature is achieved through provision of the mechanisms described in subclause 7.2.2.

NOTE: In this context, external data can be retrieved through a system component which is one service package provided by the run-time environment.

7.1.5 Access to external run-time services

This feature is achieved through provision of the mechanisms described in subclause 7.2.2.

NOTE: In this context, external calculation capability is provided by a library or process which is one service package provided by the run-time environment.

7.1.6 Computations, variable handling and control structures

This feature is achieved through provision of the mechanisms described in subclause 7.2.1.

7.2 Functional features

The MHEG-SIR is used to express:

- data processing operations;
- access to external data and functions.

7.2.1 Data processing operations

Data processing operations are provided by mechanisms that allow interchanged scripts to express:

- declaration of constructed and advanced numeric data types;
- variables and values of these types;
- instructions that perform data access or variable assignment;
- instructions that affect the script execution control flow;
- instructions that perform arithmetic, logical and comparison operators.

These mechanisms are described in clauses 9, 13 and 14.

7.2.2 Access to external data and functions

Access to external data and functions especially involve the capability to co-operate, i.e. to invoke functions, receive messages and exchange data with:

- the MHEG engine;
- the run-time environment.

Co-operation with the MHEG engine is provided by mechanisms that allow interchanged scripts to:

- invoke MHEG elementary actions;
- respond to MHEG actions targeted at the rt-script;
- express variables and values of MHEG data types.

These mechanisms are described in clause 12. They are also used to express data interchange and synchronisation either between the rt-script and the MHEG engine or between rt-scripts.

Co-operation with the run-time environment is provided by mechanisms that allow interchanged scripts to express:

- declaration of the structure of packages provided by the run-time environment;
- invocation of services provided by the run-time environment;
- reaction to messages sent by the run-time environment;
- declaration of constructed and advanced numeric data types;
- variables and values of these types.

These mechanisms are described in clause 11. They also express interchange of complex data and synchronisation between the MHEG-SIR script interpreter and processes that are part of the run-time environment.

In addition, this ETS defines the mapping of MHEG-SIR package declarations to actual run-time environment components. This is done in two steps:

- expression and use of an abstract interface specification by the MHEG-SIR: this is the IDL mapping mechanism described in clause 14. It consists of provisions for MHEG-S interchanged scripts;

- mapping between an abstract interface specification and its implementation within the run-time environment of a type of platform: this is the platform mapping specification form described in annex D. It consists of provisions for MHEG-S implementations.

7.3 Technical features

The MHEG-SIR meets the following technical requirements:

- hardware independence;
- final form representation;
- compactness;
- ease of implementation;
- interpretation efficiency;
- openness and extensibility;
- resistance to reverse engineering;
- provisions for real-time interchange;
- semantic validation for quality of service purposes;
- syntax checkability (with regard to contamination hazards);
- non-proprietary representation;
- secure script processing.

7.3.1 Hardware independence

Hardware independence of the MHEG-SIR, and therefore portability of interchanged scripts, is achieved through the definition of a virtual machine code to express interchanged scripts and the definition of a virtual machine to interpret this code. There is no requirement on the way data are represented or handled internally by MHEG-S engines.

The coded representation is based on ASN.1 encoding rules, which are hardware-independent.

The interface declarations are based on a mapping to abstract interface specifications that can be expressed using IDL, which is hardware-independent.

The capability for a component of the run-time environment of a given platform to interoperate with any conforming MHEG-S engine on this platform is guaranteed through the use of the platform mapping specification.

The capability for an MHEG-S engine implementation on a given platform to interoperate with any service provider within the run-time environment is guaranteed through the use of the platform mapping specification, provided such service providers conform to this specification.

7.3.2 Final form representation

Final form representation of interchanged scripts is achieved through:

- the use of ASN.1 for the encoding of interchanged scripts;
- a virtual machine encoding that is semantically close to a broad class of general purpose computers;
- a stack machine architecture which has an efficient instruction encoding based on implied addressing mode;
- an ordering of declarations which reduces overhead for processing of forward references;
- the appropriate sequencing of instructions within a routine.

7.3.3 Compactness

Compactness of the coded representation of interchanged scripts is achieved through several optimisations:

- the definition of a stack machine-based virtual machine code allows instructions to have few or no operands, the longest instruction taking 4 bytes to encode;
- use of bytestream coding for the routines code is used to elude the overhead induced by Type, Length and Value (TLV) coding;
- instructions are packed (i.e. do not have padding bytes) in the encoding of the routines code;
- the use of a typed stack allows dynamic polymorphism, which in turn is used to have a reduced instruction set;
- constants are used for the declaration of immediate values;
- predefined codes are defined for MHEG types, operations and messages;
- the declaration of a handler definition table is used to optimise the expression of the mapping between messages targeted at the script and routines intended to handle these messages.

7.3.4 Ease of implementation

Ease of implementation of MHEG-SIR script interpreters is achieved through:

- the definition of a reduced instruction set;
- the clear definition of a virtual machine;
- the limited number of concepts and identifiers that script interpreters need to handle;
- the formal definition of instruction semantics.

7.3.5 Interpretation efficiency

Efficiency of interpretation of interchanged scripts by MHEG-SIR script interpreters is achieved through:

- the use of a stack-based virtual machine code;
- the use of low-level instructions;
- the use of a final form representation.

7.3.6 Openness and extensibility

Openness and extensibility of the MHEG-SIR is achieved through:

- the generic definition of interfaces that can be accessed from MHEG-SIR script code;
- the capability to access MHEG objects and to invoke routines from another rt-script;
- the possibility to add new instructions to the representation without modifying the structure of interchanged scripts.

7.3.7 Resistance to reverse engineering

Resistance to reverse engineering is achieved through the use of a final form, low-level representation. This representation is for production via specialised computer tools and does not allow easy reversion to the original source code as designed by humans using scripting languages and/or authoring environments, and thus limits the risk for undue alteration of the program semantics.

7.3.8 Provisions for real-time interchange

The MHEG-SIR fits within the framework of MHEG whose general structure was designed to meet real-time interchange requirements.

The syntax of interchanged scripts is defined so as to optimise the possibility to treat the declarations it contains "on the fly".

Moreover, the use of ASN.1 encoding is used to detect errors (to some extent) while interchanging scripts in noisy network environments.

7.3.9 Semantic validation for quality of service purposes

Due to the requirements posed by this ETS on semantics, it is possible to test the behaviour of an interchanged script in order to validate its performance before it is actually used in the context of a commercial service. It is possible to build MHEG-SIR script interpreters that can serve as a reference with regard to the way conforming MHEG-S engines will behave when interpreting the interchanged scripts under test.

7.3.10 Syntax checkability (with regard to contamination hazards)

The formal definition of the MHEG-SIR syntax can be used to check its correctness and therefore prevent the interchange of pieces of code such as viruses intended to damage the receiving system in some way. It is an implementation choice as to whether syntactic and semantic checks are performed at run-time or at load-time.

7.3.11 Secure script processing

A system designer may wish to insure that faulty script execution, intentional or accidental, will have minimal impact on the delivery system, and that all access to external services can be carefully monitored. The MHEG-SIR virtual machine includes a number of features which support this goal, including:

- explicit, strongly-typed interfaces to all external services;
- a strongly-typed instruction set, so that operations and operands can be verified either by pre-processing or by run-time checking;
- no direct addressing of memory (i.e., no pointer arithmetic), preventing the possbility of spurious side-effects;
- isolation of context for each rt-script object;
- isolation of contexts defined by each call frame;
- no direct manipulation of handles, type identifiers or data identifiers.

8 Elements of the MHEG-SIR

This clause describes the main elements of the MHEG-SIR.

The entities which are declared and manipulated by MHEG-SIR interchanged scripts are:

- data types;
- data;
- functions;
- messages.

The definition of these concepts is described in subclauses 9.1 to 9.4, and the detailed structure of these declarations is described in clause 13.

8.1 Data types

Data types are used to describe the structure of:

- the script's own data (constants and variables);
- the parameters and return values of the script's routines;
- the parameters and return values of external functions;
- the parameters of messages handled by the script.

In order to allow scripts to adapt to the signature of functions that can be provided by the external environment, the MHEG-SIR allows for the definition of a wide range of types, corresponding to the IDL data types.

The encoding of data type definitions in an interchanged script is defined in annex A. This ETS imposes no requirement on the way MHEG-S engines represent these data types.

The MHEG-SIR uses three kinds of data types:

- primitive types;
- constructed types;
- predefined types.

8.1.1 Primitive types

The primitive types correspond to the IDL primitive types. This is the list of MHEG-SIR primitive types:

- void;
- boolean;
- octet;
- short;
- long;
- unsigned short;
- unsigned long;
- float;
- double;
- character;
- string;
- data identifier;
- object reference.

Primitive types have predefined type identifiers and therefore need not be declared by interchanged scripts.

8.1.1.1 The "void" type

The "void" type shall only be used to express the type of return value of a function. Functions whose type of return value is "void" do not return any data. There shall be no constants or variables of "void" type. The "void" type shall not be used in the definition of constructed types.

8.1.1.2 The "boolean" type

Data whose type is "boolean" shall have either "true" or "false" as their value. Boolean variables without an explicit initial value shall be initialised to "false".

8.1.1.3 The "octet" type

Data whose type is "octet" shall take a numeric value between 0 and 255. Octet variables without an explicit initial value shall be initialised to 0.

8.1.1.4 The "short" type

Data whose type is "short" shall take a signed integer value between -32 768 and +32 767. Short variables without an explicit initial value shall be initialised to 0.

8.1.1.5 The "long" type

Data whose type is "long" shall take a signed integer value between -2 147 483 648 and +2 147 483 647. Long variables without an explicit initial value shall be initialised to 0.

NOTE: It is to be studied whether an extra-long 64-bit integer type is needed.

8.1.1.6 The "unsigned short" type

Data whose type is "unsigned short" shall take an unsigned integer value between 0 and +65 535. Unsigned short variables without an explicit initial value shall be initialised to 0.

8.1.1.7 The "unsigned long" type

Data whose type is "unsigned long" shall take an unsigned integer value between 0 and +4 294 967 295. Unsigned long variables without an explicit initial value shall be initialised to 0.

8.1.1.8 The "float" type

Data whose type is "float" shall take a signed floating-point value whose mantissa ranges between -8 388 608 and +8 388 607 and whose exponent ranges between -128 and +127. Float variables without an explicit initial value shall be initialised to 0.

8.1.1.9 The "double" type

Data whose type is "double" shall take a signed floating-point value whose mantissa ranges between -140 737 488 355 328 and +140 737 488 355 327 and whose exponent ranges between -32768 and +32767. Double variables without an explicit initial value shall be initialised to 0.

8.1.1.10 The "character" type

Data whose type is "character" shall take a character value within the BMPString character set as defined by the Basic Multilingual Plane of ISO/IEC 10646-1 [7]. Character variables without an explicit initial value shall have no defined value.

Conforming MHEG-S engines may state that they only adopt a restricted set of characters, e.g. based on the standard collections of ISO/IEC 10646-1 [7], annex A. In this case, they shall document these adopted subsets and the level of implementation in the conformance document.

8.1.1.11 The "string" type

Data whose type is "string" shall take as their value a sequence of zero or more characters as defined by the "character" type above. String variables without an explicit initial value shall be initialised to a null string, i.e. a string of zero characters.

8.1.1.12 The "data identifier" type

Data whose type is "data identifier" shall take a signed integer value between -32 768 and +32 767 that identifies a constant, global variable, local variable or routine parameter of the script, as defined by subclause 8.7. Data identifier variables without an explicit initial value shall have no defined value.

8.1.1.13 The "object reference" type

Data whose type is "object reference" shall take as their value a handle that references an "object" on which an external function applies. Encoding of object references is defined by the platform mapping specification. Object reference values may only be provided by the external environment. There shall be no constants of "object reference" type. The "void" type shall not be used in the definition of constructed types. Object reference variables without explicit initial value shall have no defined value.

8.1.2 Constructed types

Constructed types are defined using type constructors and primitive types. This is the list of MHEG-SIR type constructors:

- sequence;
- array;
- structure;
- union;
- enumerated.

Constructed types can use as an element:

- any primitive type, except "void" and "object reference";

- any constructed type.

Unlike primitive types, an interchanged script needs to declare the definitions of the constructed types it handles.

There shall not be more than 28 672 types declared in an interchanged script.

8.1.2.1 Sequence types

Sequence types shall be defined by:

- their size (optional);
- their element type.

The size is an unsigned short value. It represents the maximum number of elements of the sequence. If the type definition specifies no size, the number of elements may be any size up to the maximum. The maximum size of any sequence is 65 535 elements.

The element type may be any primitive, constructed or predefined type. A type identifier may be used to reference the element type provided the type can be resolved, i.e. leads to no infinite recursion.

Data whose type is a defined sequence type shall take as their value an ordered list of zero or more values of the element type.

Variables of a sequence type without explicit initial value shall be initialised to a null list (sequence of zero elements).

8.1.2.2 Array types

Array types shall be defined by:

- their size;
- their element type.

The size is an unsigned short value. It represents the exact number of elements in the array.

The element type may be any primitive, constructed or predefined type. A type identifier may be used to reference the element type provided the type can be resolved, i.e. leads to no infinite recursion.

Data whose type is a defined array type shall take as their value an ordered list of values of the element type, the length of the list being specified by the size of the array.

Variables of an array type without explicit initial value shall be initialised to a list of elements whose initial value is determined by the element type.

8.1.2.3 Structure types

Structure types shall be defined by an ordered list of 1 to 256 element types.

There shall not be more than 256 elements in a structure type.

The element types may be any primitive, constructed or predefined type. A type identifier may be used to reference an element type provided the type can be resolved, i.e. leads to no infinite recursion.

Data whose type is a defined structure type shall take as their value an ordered list of values of the element type that corresponds to their rank in the type definition.

Variables of a structure type without explicit initial value shall be initialised to a list of elements whose initial value is determined by their element type.

8.1.2.4 Union types

Union types shall be defined by an ordered list of element types.

There shall not be more than 256 choices (element types) in a union type.

The element types may be any primitive, constructed or predefined type. A type identifier may be used to reference an element type provided the type can be resolved, i.e. leads to no infinite recursion.

Data whose type is a defined union type shall take as their value:

- an integer which represents the index (starting at 0) in the choice list;
- a value of the element type whose rank in the type definition is the above index.

Variables of a union type without explicit initial value shall not be initialised to any defined value.

8.1.2.5 Enumerated types

Enumerated types shall be defined by an ordered list of short integer values.

There shall not be more than 256 items (values) in an enumerated type.

Data whose type is a defined enumerated type shall take as their value an integer which represents the index (starting at 0) of the value in the enumerated type definition.

Variables of an enumerated type without explicit initial value shall be initialised to the first element in the definition.

8.1.3 Predefined types

In order to allow scripts to express manipulation of MHEG data more easily, the MHEG-API data types, as described by ISO 13522-1 [1] and ITU-T Recommendation T.177 [4], are predefined.

This means that these types, like the primitive types, have predefined type identifiers and therefore need not be declared within the script, although they are not themselves primitive types and may be expressed using type constructors.

The list of predefined types and their identifiers is given in annex C.

All types may be referenced in a unique and unambiguous way by their type identifier.

8.2 Data

The MHEG-SIR defines three kinds of data:

- immediate values;
- constants;
- variables.

All data used by an interchanged script are always of a definite data type, either primitive, constructed or predefined.

All variables and constants may be referenced in a unique and unambiguous way by their data identifier.

8.2.1 Immediate values

Immediate values are data which are not declared within the interchanged script, and can therefore only be used "immediately", i.e. at the time where they are encountered. Immediate values may be encountered in an interchanged script:

- as the value of constants;

- as the initial value of variables;
- as the operand of a "push immediate" (PUSHI) instruction.

Apart from these cases, immediate values are used in the course of the script execution through the parameter stack. This is employed to use them as parameters for either instructions or functions.

Unless the context restricts it, immediate values may be of any type except "void".

The encoding of data values in an interchanged script is defined by annex A. This ETS imposes no requirement on the way MHEG-S engines represent data values of a particular type.

8.2.2 Constants

Constants shall be declared within the interchanged script and defined by:

- a data type;
- a value of this data type.

Constants may be of any type except the following:

- object reference;
- void.

Throughout the interchanged script, constants may be referenced by their data identifier. The semantics of a reference to a constant is the same as if the value of this constant was provided instead.

There shall not be more than 4 096 constants declared in an interchanged script.

8.2.3 Variables

Variables shall be declared within the interchanged script and defined by:

- a data type;
- optionally, a value of this data type, to be taken as the initial value for this constant.

Variables may be of any type except "void".

Variables may be referenced by their data identifier. A reference to a variable may be used with two different semantics:

- "right-hand" semantics: the same as if the value of this variable was provided instead;
- "left-hand": states that this variable has to be assigned a data value.

In the latter case, the value to be assigned to the variable may be an immediate value (including a computed value), the value of a constant or the value of a variable (including the future value of a function's output parameter).

The MHEG-SIR defines two kinds of variables:

- global variables;
- local variables.

8.2.3.1 Global variables

Global variables have a scope which covers the entire interchanged script. They may be referenced by their data identifier in any part of the script. They may be assigned a new value at any time.

There shall not be more than 28 672 global variables declared in an interchanged script.

8.2.3.2 Local variables

Local variables have a scope which is restricted to the execution of the routine to which they belong. They may be referenced by their data identifier only within the code of this routine.

There are two kinds of local variables:

- local variables which are declared within the routine declaration as part of the local variable declaration;

- actual parameters of the routine, whether passed by value or by reference, which are declared within the routine declaration as part of the routine signature.

There shall not be more than 32 768 local variables declared in each routine of an interchanged script.

8.3 Functions

The MHEG-SIR defines three kinds of functions:

- routines;
- external services;
- predefined functions.

All functions have a signature (or prototype) which consists of:

- a type of return value;
- an ordered list of formal parameters defined by their type and passing mode.

All functions may be referenced in a unique and unambiguous way by their function identifier.

8.3.1 Routines

Routines are internal functions of interchanged scripts.

Routines shall be declared within the interchanged script. They consist of:

- a signature;
- local variables;
- program code.

There shall not be more than 4 096 routines declared in each routine of an interchanged script.

Execution of a routine may be triggered:

- by an explicit "call" (CALL) instruction, when the routine's function identifier is the operand of the instruction;
- upon reception of an exception during an "external call" (XCALL) instruction, when the message identifier of the received exception is mapped to the routine's function identifier by the handler definition table;
- upon peeking into the queue of received messages, when the message identifier of the received message is mapped to the routine's function identifier by the handler definition table.

Parameters may be passed to routines using two modes:

- by value: A value of the parameter type is passed to the routine;

- by reference: A data identifier referencing a variable or constant whose type is the same as the parameter type is passed to the routine.

In both cases, the value of the passed parameter becomes the value of the local variable whose index corresponds to the parameter's index.

8.3.2 Services

Services are external functions provided by the run-time environment.

Services shall be declared within the interchanged script, as part of a package declaration, by:

- their signature;
- their IDL global operation name.

There shall not be more than 256 services declared in each package of an interchanged script.

There shall not be more than 224 packages declared in an interchanged script.

A service may be called by an "external call" (XCALL) instruction.

Parameters may be passed to services using three modes:

- in: A data identifier referencing a variable or constant whose type is the same as the parameter type, is passed to the service;

- inout: A data identifier referencing a variable whose type is the same as the parameter type is passed to the service. Upon returning, the variable is updated with its new value;

- out: same as inout, however the value of the variable or constant is not used by the service.

8.3.3 Predefined functions

Predefined functions are functions provided by the MHEG engine.

In order to allow scripts to express invocation of MHEG actions more easily, the MHEG-API operations, as described by ITU-T Recommendation T.177 [4], are predefined.

This means that these functions have predefined function identifiers and therefore need not be declared within the script.

The list of predefined functions and their identifiers is given in annex C.

These functions may be called and passed parameters, using the same mechanisms as with services.

8.4 Messages

The MHEG-SIR defines two kinds of messages:

- package exceptions;
- predefined messages.

All messages have a signature (or prototype) which consists of an ordered list of formal parameters defined by their type.

All messages may be referenced in a unique and unambiguous way by their message identifier.

8.4.1 Package exceptions

Package exceptions are messages sent to the rt-script by the run-time environment, following the invocation of a service.

Package exceptions shall be declared within the interchanged script, as part of a package declaration by:

- their signature;
- their IDL global exception name.

There shall not be more than 256 exceptions declared in each package of an interchanged script.

8.4.2 Predefined messages

Predefined messages are messages which result from either:

- an exception raised as the consequence of the invocation of a predefined function;
- an MHEG action targeted at the rt-script.

In order to allow scripts to express handling of MHEG actions more easily, the messages corresponding to these exceptions and actions, as described by ISO/IEC 13522-1 [1] and ITU-T Recommendation T.177 [4], are predefined. This means that they need not be declared within the interchanged script.

The list of predefined messages and their identifiers is given in annex C.

8.5 Instructions

The program code part of routines consists of a sequence of instructions. Unlike the rest of an interchanged script, which is intended to be handled as soon as the script is prepared, instructions are only intended for execution upon activation of the rt-script, more precisely upon invocation of the routine to which they belong.

An instruction consists of one op-code (operation code) followed by zero or more operands. The number, type and encoding of operands is completely determined by the op-code.

As a rule, operands complete the instruction, but the parameter values are taken from the parameter stack.

The performance of the instruction execution unit is described in clause 10, while the precise semantics of each instruction is described in clause 14.

8.6 Identifiers

Identifiers are used to reference data throughout interchanged scripts.

8.6.1 Type identifiers

Type IDentifiers (TID) shall be encoded on two bytes as follows:

- primitive types and predefined types shall have predefined TIDs as defined by annex C;
- declared types whose index (starting at 0) in the type declaration table is X shall have (X + 1000h) as their TID.

This means that:

- TIDs between 0 and 0FFFh shall reference predefined types;
- TIDs between 1000h and 7FFFh shall reference declared types.

8.6.2 Data identifiers

Data IDentifiers (DIDs) shall be encoded on two bytes as follows:

- constants whose index (starting at 0) in the constant declaration table is X shall have X as their DID;

- global variables whose index (starting at 0) in the global variable declaration table is X shall have (X + 1000h) as their DID;

- local variables whose index (starting at 0) in the local variable declaration table is X shall have (X OR 8000h) as their DID.

This means that:

- DIDs between 0 and 0FFFh shall reference constants;
- DIDs between 1000h and 7FFFh shall reference global variables;
- DIDs between 8000h and FFFFh shall reference local variables.

8.6.3 Function identifiers

Function IDentifiers (FIDs) shall be encoded on two bytes as follows:

- routines whose index (starting at 0) in the routine declaration table is X shall have X as their FID;
- predefined functions whose index (starting at 0) in the predefined function table is X shall have (X + 1000h) as their FID;
- services whose index (starting at 0) in a package declaration is X and whose package index in the package declaration table is Y (starting at 0) shall have (((Y+32) << 16) + X) as their FID.

This means that:

- FIDs between 0 and 0FFFh shall reference routines;
- FIDs between 1000h and 1FFFh shall reference predefined functions;
- FIDs between 2000h and FFFFh shall reference services.

8.6.4 Message identifiers

Message IDentifiers (MIDs) shall be encoded on two bytes as follows:

- predefined messages whose index (starting at 0) in the predefined message table is X shall have X as their MID;
- exceptions whose index (starting at 0) in a package declaration table is X and whose package index (starting at 0) in the package declaration table is Y shall have (((Y+32) << 16) + X) as their MID.

This means that:

- MIDs between 0 and 1FFFh shall reference predefined messages;
- MIDs between 2000h and FFFFh shall reference package exceptions.

8.7 Type matching

Two data (i.e. variables, constants, elements of the parameter stack) have formally matching types if and only if they have the same type identifier.

The MHEG-SIR does not allow type redefinition, i.e. no two type identifiers shall have the same type structure. Therefore, two types match only if they have the same identifier. No other rules for type matching are defined.

N-level type definition is forbidden. Each structured type may be defined using already defined types at each level of nesting.

8.8 Value matching

Values of matching types shall be equal only if:

- they are of a primitive or enumerated type, they are identical;
- they are of a structure, sequence or array type, every element of one list is equal to the element of the same rank in the other list;
- they are of a union type, their tags are identical and their values are equal to each other.

9 The MHEG-SIR virtual machine

This clause presents the MHEG-SIR virtual machine, i.e. the execution model for the MHEG-SIR pseudo-code.

The MHEG-SIR virtual machine is a set of logical, abstract components. The description of the MHEG-SIR virtual machine is intended for clarification of the operational semantics of the MHEG-SIR code.

An MHEG-S engine shall have the same interpretation behaviour for MHEG-SIR code as the described virtual machine. It shall interpret MHEG-SIR declarations and instructions so as to produce similar external effects in all respects.

However, this implies no requirements on the technology or organisation that may actually be used to implement an MHEG-S engine. An actual script interpreter need not be designed as described by the virtual machine, as long as it provides equivalent functionality.

The MHEG-SIR virtual machine consists of:

- memory areas;
- processing units.

MHEG-S engines may run several rt-scripts at the same time. In this case, the MHEG-S engine should maintain a separate run-time context for each rt-script.

NOTE: The MHEG-SIR virtual machine is single threaded. Multi-threaded applications can be achieved by associating each thread with a separate rt-script.

9.1 Structures and notations

A table T consists of an array of homogeneous entries T[i] that can be accessed via their index i. Homogeneous means that they have the same structure, but not necessarily the same size. Entries consist of one or several fields accessed as T[i].fld. Some entries may be void. Indices are MHEG-SIR identifiers, i.e. consecutive numeric values taken in a given range. The underlying access mechanism (sequential indexing, direct access, hashcoding) is not specified.

A stack S consists of an array of homogeneous entries. Only the top entry (last entered) can be accessed at any time, through the stack pointer p, as S[p]. Only the entries below the stack pointer are maintained. This means that when the pointer is decremented (p--), the top element of the stack is lost, and the next element becomes the top of the stack.

The representation of the structures and data is implementation-dependent. Although it is possible for script interpreters to represent each value of a data type with a minimum number of bytes, there is no requirement to do so. Table 1 hereafter states this minimum number:

Table 1: Minimum number of bytes to represent values

Type	Minimum number of bytes to represent a value of the type
boolean	1
octet	1
short	2
long	4
unsigned short	2
unsigned long	4
float	4
double	8
character	2 (1 for restricted character sets)
string	character size x string length +1
data identifier	2
object reference	implementation-dependent
structure	sum of the sizes of the element types
sequence	size of element type x sequence length
union	size of the largest element type
enumerated	2
array	size of element type x array dimension
type identifier	2
function identifier	2
message identifier	2
package identifier	1

The notation does not distinguish between fixed-length values and values that are likely to be stored on the heap. GT[i].val refers to the value even though it is actually stored as a handle to an area of the heap.

Execution semantics are expressed using a C-like syntax.

9.2 Memory areas

In the MHEG-SIR virtual machine, memory areas are used to hold all the necessary information used to interpret a particular interchanged script.

Some of these memory areas are completely filled in upon preparation of the script, while others are modified during execution of the script. Some are specific to one rt-script, while the others are common to the different rt-script instances.

The MHEG-SIR virtual machine holds four memory areas:

- global data area;
- code area;
- dynamic memory area;
- registers.

9.2.1 Global data area

The global data area is used to store the definitions and values of the script's global data. The global data area is filled upon preparation of the script. Once this is done, the size of the global data and structure remains unmodified thereafter.

The global data area consists of:

- the Type definition Table (TT);
- the Constant Table (CT);
- the Global Variable table (GT).

9.2.1.1 The type definition table

The Type definition Table is used to map all the defined types of the script, represented by Type IDentifiers, to their description:

- TT[TID].val: description of the type.

The TT is filled in upon preparation of the script object, and should not be modified afterwards, until the script object is destroyed. The TT may be shared between the different rt-script instances.

NOTE: The representation used for the type description is unspecified; however, it should allow easy checking of whether a value belongs to a type.

Whenever intermediate constructed types are used in the description, an unassigned TID should be created and inserted in the TT; this should facilitate type matching operations especially in comparison operators and instructions performing variable element assignment and access.

9.2.1.2 The constant table

The Constant Table (CT) is used to map all the script's constants, represented by data identifiers, to their type and value:

- CT[Data IDentifier (DID)].TID: type of the constant (expressed as a type identifier);
- CT[DID].val: value of the constant (depending on its type).

The CT is filled in upon preparation of the script object, and is static read-only until the script object is destroyed. The CT may be shared between the different rt-script instances.

When CT[DID].TID represents a constructed type, CT[DID].val should have the following structure:

- CT[DID].val.nbe: number of elements at this level;
- CT[DID].val[i].TID: type of the element of index i (this TID may have been created by the script interpreter as specified above);
- CT[DID].val[i].val: value of the element of index i (this field may recursively have a constructed value structure).

9.2.1.3 The global variable table

The global variable table is used to map all the script's global variables, represented by data identifiers, to their type and current value:

- GT[DID].TID: type of the global variable (expressed as a type identifier);
- GT[DID].val: current value of the global variable (depending on its type).

The global variable table is initialised upon preparation of the script. Its GT[DID].val fields should afterwards be modified every time a global variable is assigned by the execution of a variable assignment instruction.

When GT[DID].TID represents a constructed type, GT[DID].val should have the following structure:

- GT[DID].val.nbe: number of elements at this level;
- GT[DID].val[i].TID: type of the element of index i (this TID may have been created by the script interpreter as specified above);
- GT[DID].val[i].val: value of the element of index i (this field may recursively have a constructed value structure).

9.2.2 Code area

The code area is used to store the addresses and program code of the script's functions. The code area is filled upon preparation of the script object. The code area's size and structure should remain static until the script object is destroyed.

The code area consists of:

- the Routine definition Table (RT);
- the Package definition Table (PT);
- the Service definition Table (ST);
- the Exception definition Table (XT);
- the Handler definition Table (HT);
- the program code area, consisting of the sequence of instructions of each routine.

9.2.2.1 The routine definition table

The Routine definition table is used to map all the script's routines, represented by function identifiers, to their signature description, their local variable declaration and their program code. The RT contains:

- RT[FID].TID: type of return value (expressed as a type identifier);

- RT[FID].nbp: number of parameters;

- RT[FID].sig: signature description, where:

  a) RT[FID].sig[i].TID: type (expressed as a type identifier) of the ith parameter;

  b) RT[FID].sig[i].mod: passing mode (value or reference) of the ith parameter.

- RT[FID].Local variable Table (LT): declaration of the routine's local variables (whose nbp first elements are the actual parameters of the routine);

- RT[FID].Instruction Pointer (IP): pointer to the first instruction in the routine code.

The RT is filled in upon preparation of the script, and need not be modified afterwards. The RT may be shared between the different rt-script instances.

NOTE: Local variables used to hold parameters passed by reference should have "data identifier" as their type, while local variables used to hold parameters passed by value should have the same type as in the signature description for the corresponding parameter.

9.2.2.2 The package definition table

The package definition table is used to map all the script's defined packages, represented by package numbers (PIDs) as declared by the MHEG-SIR package declaration table, to package names and additional information:

- PT[PID].name: name of the package;
- PT[PID].sts: current status of the package;
- PT[PID].nbf: number of services in the package;
- PT[PID].nbm: number of exceptions defined by the package;
- PT[PID].rte: may be used to store platform-specific information allowing addressing of the package within the run-time environment.

The PT is filled in upon preparation of the script. Only its PT[PID].sts fields may be modified afterwards, each time a package is loaded or unloaded from the run-time environment, open or closed. The PT may be shared between the different rt-script instances.

9.2.2.3 The service definition table

The service definition table is used to map all the script's defined external services, represented by MHEG-SIR function identifiers, to their signature description and to their IDL global operation name:

- ST[FID].TID: type of return value (expressed as a type identifier);

- ST[FID].syn: calling mode (synchronous, asynchronous);

- ST[FID].nbpar: number of parameters;

- ST[FID].sig: signature description, where:

  a) ST[FID].sig[i].TID: type (expressed as a type identifier) of the ith parameter;

  b) ST[FID].sig[i].mod: passing mode (in, inout or out) of the ith parameter.

- ST[FID].name: the IDL global name of the operation which the service invokes;

- ST[FID].rte: may be used to store platform-specific information allowing addressing of the service within the run-time environment.

The IDL platform-specific mapping specification is used to map XT[MID].name to a platform-specific name.

The table is filled in upon preparation of the script, and need not be modified afterwards. It may be shared between the different rt-script instances.

9.2.2.4 The exception definition table

The eXception definition table is used to map all the interchanged script's defined messages, represented by message identifiers, to their signature description and their IDL global exception name:

- XT[MID].name: the IDL global name of the exception which causes the message;
- XT[MID].rte: may be used to store platform-specific information allowing addressing of the exception within the run-time environment.

The IDL platform-specific mapping specification is used to map XT[MID].name to a platform-specific name.

The XT is filled in upon preparation of the script, and need not be modified afterwards. It may be shared between the different rt-script instances.

9.2.2.5 The handler definition table

The handler definition table is used to map messages, represented by message identifiers, to routines represented by function identifiers:

- HT[MID].FID: identifier of routine to invoke for handling the message.

The handler definition table is filled in upon preparation of the script, and need not be modified afterwards. It may be shared between the different rt-script instances.

The signature of the routine which maps to a message matches the signature of this message. Matching between the signatures is checked at load time, where non-matching entries are rejected.

The handler definition table is used by the script execution unit. When the message queue contains a message, the routine that corresponds to the message is invoked, with the message parameters as its parameters.

9.2.2.6 The program code area

An instruction consists of one 1 byte op-code followed by 0 to 3 operand bytes. The op-code completely determines the number and length of its operands, according to the instructions table. Both op-codes and operands are coded in an optimised fashion so as to ease switching.

It is possible (especially for 32-bit machines) to align instructions, i.e. to insert padding bytes in order to represent each instruction on 4 bytes; this makes it easy to increment the program counter. It is also

possible to pack instructions instead of aligning them, and to determine the number of bytes to increment at interpretation time.

The program code area is filled in upon preparation of the script, and is not modified during execution. It may be shared between the different rt-script instances.

9.2.3 The dynamic memory areas

The dynamic memory area is used to represent the current execution context of the rt-script.

The dynamic memory area consists of:

- the Calling Stack (CS);
- the Parameter Stack (PS);
- the Message Queue (MQ);
- the heap area.

9.2.3.1 The calling stack

The Calling Stack contains the current invocation context.

The CS is an array of call frames. Every frame corresponds to an active function invocation (routine, external function or MHEG action). Frames are stored on the CS in order of invocation. The top frame on the CS is the current execution context.

Every frame contains:

- CS[i].FID: function identifier of the callee;
- CS[i].IP: pointer to the instruction to return to after the function has been executed;
- CS[i].LT: local variable table of the callee;
- CS[i].Stack Pointer (SP): pointer to the top of the parameter stack at the time of calling.

The LT has the structure of a variable table:

- CS[i].LT[DID].TID: type identifier of the variable whose identifier is DID;
- CS[i].LT[DID].val: value of the variable.

The first entries of the LT are the parameters passed to the function. For an external function, the LT contains only these parameters.

The calling stack is modified by certain control flow instructions. Initially the call stack is empty and the Function Pointer (FP) register is set to the first available position minus one. When a function is invoked, a frame describing this call is pushed onto the calling stack. When a function is returned from, this frame is popped from the calling stack. The address of the top frame of the calling stack is stored in the FP register.

9.2.3.2 The parameter stack

The Parameter Stack stores the parameters and return values of instructions. The parameter stack is an array of data values. The type of the data value is implied by the operation sequence that created the value on the stack.

The PS is used by the MHEG-SIR instruction execution unit. Initially the PS is empty and the SP register points to the first available position on the PS minus one. It is modified by most instructions (arithmetic operators, logical operators, comparison operations, stack manipulation, variable assignment, conditional jumps, calls). When an instruction is executed, it pops its parameters from the PS, and pushes its return value back onto the PS. The address of the top frame of the PS is stored in the SP register.

9.2.3.3 The message queue

The Message Queue is used for buffering the messages which are received by the script interpreter. Every item in the queue contains:

- MQ[i].MID: message id;
- MQ[i].LT: list of message parameters.

The LT field of a message queue item has the structure of a variable table:

- MQ[i].LT[j].TID: type identifier of the jth parameter;
- MQ[i].LT[j].val: value of the jth parameter.

Messages are inserted into the message queue by the script interpreter asynchronously as they are generated in the external environment. The message queue is processed by the script code execution unit when:

- the execution of a routine finishes, without any routine to return to;
- the script explicitly yields control via a YIELD instruction.

NOTE: It is to be studied whether an interrupt-style message processing mechanism is needed.

The start of the message queue (next message to pop) is stored in the Queue Pointer (QP) register. Initially the message queue is empty and QP is set to the null value.

9.2.3.4 The heap area

The heap area is used to store dynamically created variables of any type. In this case, the GT and LT store a heap handle instead of the data itself. The handle is an opaque type whose internal representation is implementation dependent. The application is responsible for explicit allocation and deallocation of all dynamically allocated storage.

9.2.4 Registers

Registers hold specific state of the virtual machine and may be frequently modified during the execution of the script.

The registers maintained by the MHEG-SIR virtual machine are:

- the IP or program counter;
- the FP;
- the SP;
- the QP.
- the Instruction Register (IR);
- the Error Register (ER);

The representation of data held by pointer registers is implementation-dependent. The script interpreter maintains one set of virtual machine registers for each available rt script.

9.2.4.1 The instruction pointer register

The IP register holds the address of the next instruction to be executed. This register is modified by the script code execution unit and by the MHEG-SIR instruction execution unit as part of the execution of instructions.

9.2.4.2 The instruction register

The IR holds the code for the instruction which is currently executing. This register is updated by the script code execution unit each time a new instruction is loaded, and accessed by the MHEG-SIR instruction execution unit.

NOTE: The IR need not be more than 4 bytes long, but its actual size depends on the implementation.

9.2.4.3 The error register

The ER contains the code of the last error encountered during execution of an instruction. This register is updated by the MHEG-SIR instruction execution unit, every time it encounters an error. The NULL (0) value means that no error has been encountered during the execution of the script.

Error codes are predefined. Which error codes can be raised by each instruction is defined in clause 14.

When the script interpreter realises that the ER is set to a non-null value, it makes this information available to the MHEG engine by setting the "termination status" of the rt-script to "error".

9.2.4.4 The stack pointer register

The SP register points to the top of the PS. The value of this register is incremented by the MHEG-SIR instruction execution unit every time it pushes data onto the PS, and decremented by the MHEG-SIR instruction execution unit every time it pops data off the PS.

9.2.4.5 The function pointer register

The FP register points to the top frame of the calling stack. The value of this register is incremented by the MHEG-SIR instruction execution unit every time a function is called, and decremented every time a function is returned from.

9.2.4.6 The queue pointer register

The QP register points to the next message to be removed from the message queue. The value of this register is decremented by the script interpreter each time a message is removed.

9.3 Processing units

This subclause describes the MHEG-SIR virtual machine's flow of control and the semantics of instructions.

9.3.1 Mh-script initialisation

When the MHEG "prepare" action targeted at a script object is triggered, the action interpreter should request the access agent to retrieve the interchanged script, then request the script interpreter to parse it. The script interpreter should then:

- parse the declarations part and initialize the CT, GT, TT, RT, ST, PT, XT, HT;
- parse the structure of the instructions part to fill in the program code area;
- load packages, establishing static links with them according to the platform mapping specification (e.g. checking their presence on the platform) and completing the rte fields within PT, ST and XT entries;
- place the script object to "available" (prepared) status.

NOTE: The semantics of package loading should be defined by the platform mapping specification. The MHEG-S engine may take the responsibility to optimise its resource management strategy, e.g. by unloading packages temporarily in order to release memory.

9.3.2 Rt-script initialisation

When the MHEG engine "new" action targeted at an rt-script is triggered, the action interpreter should consult the mh-object manager, then request the script interpreter to initialise the rt-script. The script interpreter should then:

- build an environment for the rt-script that consists of the GT and a set of registers;
- open packages, establishing dynamic links with them according to the platform mapping specification;
- initialise all registers to NULL values.

NOTE 1: The semantics of package opening should be defined by the platform mapping specification. The MHEG-S engine may take the responsibility to optimise its resource management strategy, e.g. by closing packages temporarily in order to release memory.

NOTE 2: A detailed discussion of "script object life-cycle" will be added to this section as well as subclause 12.3.

9.3.3 Message reception

Messages received by the script interpreter may be:

- messages corresponding to the triggering of MHEG elementary actions;

- messages corresponding to the occurrence of a exception raised by either an operation of the run-time environment or by an MHEG-API operation.

9.3.3.1 Elementary action

When the action interpreter transmits an MHEG elementary action targeted at an rt-script to the script interpreter, the script interpreter should:

- insert the message into the message queue of the rt-script's environment;
- give control to the script code execution unit for this rt-script.

9.3.3.2 Exception

When a message coming from either the action interpreter or the run-time environment is directed at the MHEG-S engine, and if the MHEG-S engine determines that this message actually corresponds to an exception raised by the MHEG-API or the run-time environment as a consequence of the invocation of an operation resulting from an XCALL instruction, the script interpreter should:

- parse the exception's parameters and construct an internal structure consisting of the message identifier of the exception followed by its actual parameters;

- if the invoked operation is synchronous, terminate the XCALL instruction (therefore popping its frame from the calling stack) and immediately afterwards trigger the routine corresponding to the exception's message identifier, with the exception's parameters as its actual parameters. The effect shall be the same as if this routine had been invoked by a CALL instruction;

- otherwise, insert the received message into the message queue.

9.3.4 Script code execution unit

When given control, the script code execution unit should perform as follows:

```
script-code-execution-unit ()
{
FID fid = NULL;
if (IP == NULL)                         // no next instruction
{
while (fid == NULL)
{
if (QP == NULL) then exit;          // yield control
fid= HT[MQ[QP].MID].FID;                // find handler for message
if (fid == NULL)                        // no handler found
then QP--;                              // pop message queue
else
{
IP = RT[fid].IP;                        // branch to start of routine
FP++;                                   // stack routine call
CS[FP].FID = fid;
CS[FP].IP = NULL;
CS[FP].LT = MQ[QP].LT;
CS[FP].SP = SP;
QP--;                                   // pop message queue
}
}
}
// endif

while (IP != NULL):
{
IR = *IP++;                             // load next instruction and increment program counter
instruction-execution-unit();           // give control to MHEG-SIR instruction execution unit
}
// endwhile
return;                                 // yield control to script interpreter
}
```

9.3.5 MHEG-SIR instruction execution unit

When given control, the MHEG-SIR instruction execution unit should decode the op-code contained in the first byte of the IR, perform as described by clause 14 in the interpretation of the instruction corresponding to this op-code, then give control back to the script code execution unit.

The instruction execution unit pops from the parameter stack those parameters which are used for performing the instruction (if any). It pushes on the parameter stack those parameters which are the result of the instruction (if any).

Subclause 13.2 gives an instruction table that summarises the effects of the instructions on the various elements of the MHEG-SIR virtual machine as defined by this subclause.

10 Provisions for run-time environment access

This clause describes the mechanisms defined by this ETS to make it possible for rt-scripts to access and interchange data with external functions provided by the run-time environment on the execution platform.

10.1 General model

To allow its use by an interchanged script, the interface that an external piece of software available in the run-time environment is able to provide needs to be declared in the interchanged script as part of its package declaration.

NOTE: Further study will determine the need, advantages, and drawbacks of introducing an alternative way to describe packages by transmitting them in "utility" scripts (containing only packages) that would then be referenced by scripts containing no packages. This would allow the separation of package declarations from script object code. In this case:

- utility scripts would need to be available (prepared) first, otherwise an error condition is raised;
- scripts could not proceed to "available" status unless the appropriate declarations are available;
- scripts without packages would consider "included" packages as if they were their own;
- the utility script would be referenced through an MHEG object reference;
- a script would only reference one utility script;
- type matching between private types and IDL parameter types would be needed.

A package declaration describes a set of services, i.e. external functions, by their signature, i.e. the type and passing modes of each service's parameters.

This ETS specifies how calling external functions, passing parameters, getting back return values and handling exceptions shall be expressed within interchanged scripts.

This ETS also specifies how these expressions shall be interpreted by MHEG-S engines.

This ETS also deals with interchange (i.e. function call, parameters passing, return value retrieval and exception handling) between an MHEG-S engine and the run-time environment. In this purpose, this ETS contains provisions for specifying how access to these functions should be provided to MHEG-S engines by external pieces of software. Such a convention, called a platform mapping specification, is dependent on the run-time platform.

Platform mapping specifications conforming to the provisions of this ETS need to be registered to ensure the interoperability of run-time environment services with any compliant MHEG-S engine on this platform. In this case, MHEG-S engines need to conform to this platform mapping specification in order to access run-time environment services.

Implementations of MHEG-S engines shall document in their conformance document the platform mapping specification(s) to which they conform.

NOTE: If an existing piece of software does not comply with the platform mapping specification, it will need to be embedded into a form of "glue" interface allowing translation of its own interface conventions into those specified by the platform mapping specification.

10.2 Declaration of IDL interfaces

The interface of an external piece of software intended for use by an interchanged script may contain:

- operation declarations;
- exception declarations;
- type declarations.

Types shall be declared in the type declaration of this interchanged script.

Operations and exceptions shall be declared in the package declaration of this interchanged script. This package declaration shall be assigned a package identifier and contain:

- the package name;
- a set of service descriptions;
- a set of exception descriptions.

Service descriptions shall be assigned an identifier and shall contain:

- the name of the operation;
- the function signature, i.e. the type and passing mode of each parameter.

Exception descriptions shall be assigned a message identifier and shall contain:

- the name of the exception;
- the exception signature, i.e. the type and passing mode of each parameter.

Identifiers (package identifiers, type identifiers, function identifiers) are used by MHEG-SIR scripts to refer to types and functions. A function identifier for an external operation can be built from a package identifier and the index of the service declaration in this package, while a message identifier for an external exception can be built from a package identifier and the index of the exception declaration in this package.

Names (package names, operation names, exception names) shall be used by the script interpreter to link with the actual implementation of the external piece of software.

An MHEG-SIR package declaration lies at the same abstraction level as an IDL specification. This ETS defines the rules for mapping an IDL specification into a package declaration. Clause 15 shows how:

- an IDL data type description shall be mapped to an MHEG-SIR data type description;
- an IDL operation description shall be mapped to an MHEG-SIR service description;
- an IDL exception description shall be mapped to an MHEG-SIR exception description.

10.3 Invocation of IDL operations in an MHEG-SIR program

A service described in a package declaration shall be invoked from an MHEG-SIR program as follows:

- variables of expected types corresponding to the returned value (if any) and to each parameter shall be declared within the interchanged script;

- the program shall assign those variables which correspond to input or input/output parameters;

- the program shall push onto the stack the data identifiers of all these variables in right-to-left order (the identifier of the variable corresponding to the returned value is pushed first, the identifier of the variable corresponding to the first parameter is pushed last);

- the program shall invoke the operation using an external call (XCALL) instruction whose operands are the function identifier of the called operation;

- the program shall exploit the function results through the use of the variables corresponding to the returned value, the input/output parameters and the output parameters.

10.4 Handling of IDL exceptions in an MHEG-SIR program

An exception described in a package declaration shall be handled by an MHEG-SIR program as follows:

- variables of expected types corresponding to each parameter shall be declared within the interchanged script;

- a routine whose parameters correspond to the exception shall be declared within the routine declaration part of the interchanged script;

- the mapping between the identifiers of this handling routine and the exception shall be declared in the handler declaration part of the interchanged script.

10.5 Invocation of IDL operations by an MHEG-S engine

When an interchanged script expresses invocation of an IDL operation as described in subclause 11.3, the script interpreter shall behave as described by the semantics of the XCALL instruction in clause 14. As part of this performance, it shall interpret the mechanisms described in subclause 11.3 in translating them into the run-time environment access mechanisms as defined by the platform mapping specifications.

NOTE: For example, an MHEG-S engine may translate a variable identifier pushed onto the stack as a service parameter into either a value or a real memory address to be passed to the external piece of software that provides the service.

10.6 Handling of IDL exceptions by an MHEG-S engine

When an exception is raised by an external service, this will result in a message being transmitted to the MHEG-S engine according to the run-time environment access mechanisms defined by the platform mapping specifications.

The script interpreter shall then behave as described in subclause 10.3.3.2.

10.7 Platform mapping specifications

A platform mapping specification shall contain:

- the description of the platform to which the specification applies;

- the procedure that MHEG-S engines should use to check the availability of a given package within the run-time environment;

- the procedure that MHEG-S engines should use to make the operations of a given package accessible to an rt-script;

- the procedure that MHEG-S engines should use to unload a package;

- the procedure that MHEG-S engines should use to invoke a given operation;

- the way MHEG-S engines should encode the value of input or input/output parameters of an operation;

- the way MHEG-S engines should decode the value of output or input/output parameters of an operation;

- the procedure that MHEG-S engines should use to pass input, input/output and output parameters to an operation;

- the procedure that MHEG-S engines should use to retrieve the return value of an operation;

- the procedure that MHEG-S engines should use to retrieve exceptions raised by an operation;

- the way MHEG-S engines should decode the value of parameters of an exception raised by an operation.

The contents of the platform mapping specification are described in annex D.

11 Provisions for MHEG object manipulation

This clause describes the mechanisms defined by this ETS to make it possible for rt-scripts to manipulate MHEG objects.

11.1 Invoking MHEG actions

The MHEG-SIR is used to express invocation of MHEG actions as defined by ETR 225 [9] (MHEG API).

The MHEG-API is defined using IDL. The mapping from an IDL definition to an MHEG-SIR package declaration and type declaration is defined in clause 14. The MHEG-API package is however considered as a predefined one. Its declaration shall not be included explicitly in interchanged scripts. The mapping mechanism is therefore similar to the external function declaration mechanism described in clause 11, except for the fact that the IDL types and operations defined by the MHEG-API need not be declared as part of the MHEG-SIR code, but are instead dealt with as predefined types and predefined external functions.

The mechanism used to invoke an MHEG action is similar to invocation of a service provided by the run-time environment. An XCALL instruction is used. Types defined in the MHEG-API package are referred to using a predefined type identifier. Functions described in the MHEG-API package are referred to using a predefined function identifier.

11.1.1 Sending messages to other scripts

Within an MHEG-SIR program, any kind of MHEG message may be sent to the MHEG engine using the MHEG-API. This includes sending messages targeted at mh-scripts or rt-scripts.

NOTE: For example, an rt-script may monitor the execution of another script through the use of MHEG-API operations corresponding to the "prepare", "new", "run", "get termination status", "delete", "destroy" MHEG actions.

11.1.2 Synchronisation with MHEG objects

Within an MHEG-SIR program, synchronisation of the rt-script with other MHEG objects (including other rt-scripts) may be done through the use of the MHEG-API operations corresponding to the "set data" and "get data" MHEG action. MHEG content objects embedding generic values may be used to constitute a shared memory area between MHEG objects.

Waiting for a signal from another script may be translated by a loop including a call to the MHEG-API operation corresponding to the "get data" MHEG action until the expected value is retrieved.

Generating a signal may be translated by a call to the MHEG-API operation corresponding to the "set data" MHEG action.

11.2 Receiving MHEG messages

The MHEG-SIR is used to express handling of messages coming from the MHEG engine. These messages may be of the following types:

- "return" actions;
- MHEG actions whose target is the MHEG script object (mh-object);
- MHEG actions whose target is the MHEG rt-script;
- MHEG API exceptions.

11.2.1 Return actions

Within an MHEG-SIR program, "return" actions are processed through the use of the MHEG-API. The "getReturnability" operation gives the list of available notifications while the "getNotification" operation retrieves a notification.

11.2.2 MHEG actions targeted at an mh-script

The MHEG actions that can be targeted to an mh-script are "prepare", "destroy" and "get preparation status". These actions should be handled by the script interpreter as described in clause 10, with the default effect specified in subclause 12.3; the MHEG-SIR does not provide mechanisms to express specific handling for these actions.

11.2.3 MHEG actions targeted at an rt-script

The MHEG actions that can be targeted to an rt-script are "set parameters", "get termination status", "new", "delete", "get availability status", "run", "stop", "set global behaviour", "get global behaviour", "set alias" and "get alias".

These actions should be handled by the script interpreter as described in clause 10, with the default effect specified in subclause 12.4 below.

In addition, the MHEG-SIR provides mechanisms allowing a script to express specific handling for some of these actions ("set parameters", "new", "delete", "run", "stop"). These MHEG actions targeted at an rt-script have predefined message identifiers. The handler declaration of the interchanged script is used to map these actions to routines. When the action is triggered, the script interpreter shall push the message into the message queue; when the message queue is afterwards looked up at, the script interpreter shall invoke the routine to which the message identifier is mapped, with the corresponding parameters.

11.2.4 MHEG-API exceptions

The MHEG-API exceptions are considered as messages which are sent to the script interpreter as the result of invoking an MHEG-API operation. These exceptions have predefined message identifiers. The script interpreter shall process these messages in the same way as it would process an exception coming from the run-time environment, as described in subclause 10.3.3.2.

11.3 Effect of MHEG actions

11.3.1 Prepare

When a "prepare" action is targeted at an mh-script, the script interpreter should perform the mh-script initialisation operations as described in subclause 10.3.1.

11.3.2 New

When a "new" action is targeted at an rt-script, the script interpreter should perform the rt-script initialisation operations as described in subclause 10.3.2. It should then invoke the routine mapped to the "new" predefined message identifier in the handler table, if this mapping exists.

11.3.3 Run

When a "run" action is targeted at an rt-script, the script interpreter should invoke the routine mapped to the "run" predefined message identifier in the handler table. If such a mapping does not exist, the first routine (whose function identifier is 0) shall be invoked, provided this routine either has no parameters or only has parameters passed by value whose type is such that they may be assigned a default value. If the first routine cannot be invoked, an exception shall be raised.

11.3.4 Set parameters

When a "set parameters" action is targeted at an rt-script, the script interpreter should invoke the routine mapped to the "set parameters" predefined message identifier in the handler table, if such a mapping exists.

NOTE: This action may be used to assign variables which will be used afterwards by the routine handling the "run" action.

11.3.5 Stop

When a "stop" action is targeted at an rt-script, the script interpreter should invoke the routine mapped to the "stop" predefined message identifier in the handler table, if such a mapping exists.

11.3.6 Delete

When a "delete" action is targeted at an rt-script, the script interpreter should invoke the routine mapped to the "delete" predefined message identifier in the handler table, if such a mapping exists.

The script interpreter should then clean up the rt-script-specific memory areas in order to restore a state that will enable the rt-script initialisation operations to be performed again as described in subclause 10.3.2.

NOTE: This should include closing the link between packages and this rt-script.

11.3.7 Destroy

When a "destroy" action is targeted at an mh-script, the script interpreter should clean up all memory areas related to this script and restore a state that will enable the mh-script initialisation operations to be performed again as described in subclause 10.3.1.

NOTE: This should include unloading packages, unless of course they are used by other mh-scripts.

12 MHEG-SIR declarations

This clause describes the structure of interchanged scripts. This clause also describes the way the virtual machine deals with parsing of an interchanged script.

An interchanged script shall consist of:

- a sequence of type declarations;
- a sequence of constant declarations;
- a sequence of global variable declarations;
- a sequence of package declarations;
- a sequence of message handler declarations;
- a sequence of routine declarations.

```
InterchangedScript       ::=    TypeDeclaration*
                                ConstantDeclaration*
                                GlobalVariableDeclaration*
                                PackageDeclaration*
                                HandlerDeclaration*
                                RoutineDeclaration*
```

12.1 Type declaration

Type declarations shall be used to describe the types of the interchanged script.

A type declaration shall consist of:

- a type identifier (optional);
- a type description.

```
TypeDeclaration      ::=       TypeIdentifier?
                               TypeDescription
```

12.1.1 Type identifier

Type IDentifiers (TIDs) are used to reference the type description throughout the interchanged script.

The TID shall be a positive integer within the range allowed for declared types. It shall correspond to the maximum number of predefined types incremented by the index (starting from 0) of the declaration in the type declarations part.

If the TID is not provided, it shall be computed by the script parser.

```
TypeIdentifier    ::=          INTEGER
```

12.1.2 Type description

Type descriptions describe the structure of a declared type.

The type description shall be either:

- a TID;
- an enumerated description;
- a sequence description;
- an array description;
- a structure description;
- a union description.

```
TypeDescription      ::=       TypeIdentifier
              /                EnumeratedDescription
              /                SequenceDescription
              /                ArrayDescription
              /                StructureDescription
              /                UnionDescription
```

If the type description is a TID, the type description of the identified type matches the type description for the currently declared type.

12.1.2.1 Enumerated description

An enumerated description shall consist of a sequence of integer values which compose the enumeration.

```
EnumeratedDescription ::=      INTEGER*          //           List of values
```

12.1.2.2 Sequence description

A sequence description shall consist of:

- an integer (optional);
- a type description.

```
SequenceDescription  ::=       INTEGER?                //     Sequence (max) size
                               TypeDescription
```

The integer represents the maximum size of the sequence; if it is not provided, the sequence is not bounded.

The type description represents the description of the type of element of the sequence.

12.1.2.3 Array description

An array description shall consist of:

- an integer;
- a type description.

```
ArrayDescription     ::=       INTEGER                 //     Array dimension
                               TypeDescription
```

The integer represents the size of the array.

The type description represents the description of the type of element of the array.

12.1.2.4 Structure description

A structure description shall consist of a sequence of type descriptions.

```
StructureDescription ::=       TypeDescription+
```

Each type description represents the description of one of the fields of the structure.

12.1.2.5 Union description

A union description shall consist of a sequence of one or more type descriptions.

```
UnionDescription     ::=       TypeDescription+
```

Each type description represents the description of one of the choices of the union.

12.2 Constant declaration

Constant declarations shall be used to describe the types and values of the constants of the interchanged script.

A constant declaration shall consist of:

- a data identifier (optional);
- a type identifier;
- a constant value.

```
ConstantDeclaration  ::=       DataIdentifier?
                               TypeIdentifier
                               ConstantValue
```

12.2.1 Data identifier

Data IDentifiers (DID) are used to reference data throughout the interchanged script.

The DID shall be a positive integer within the range allowed for constants. It shall correspond to the index (starting from 0) of the declaration in the constant declarations part.

If the DID is not provided, it shall be computed by the script parser.

```
DataIdentifier       ::=       INTEGER
```

12.2.2 Type identifier

The TID represents the type to which the value of the constant belongs.

12.2.3 Constant value

The constant value represents the value that the constant will correspond to throughout the script.

If the type of the constant matches a primitive type, the constant value shall consist of an immediate value expressed in this type.

If the type of the constant matches an enumerated type, the constant value shall consist of an immediate octet value representing the index (starting from 0) of the value of the constant in the enumeration.

If the type of the constant matches a structure type, the constant value shall consist of a sequence of constant values, whose length is the same as the number of elements in the structure; each of these values shall be of a type matching the corresponding element type in the structure description.

If the type of the constant matches a sequence or array type, the constant value shall consist of a sequence of constant values, whose length is less or equal to the size of the sequence type, or exactly equal to the size of the array type, and whose type match the element type of the sequence or array description.

If the type of the constant matches a union type, the constant value shall consist of an integer representing the index (starting from 0) of the choice in the union, and a constant value whose type shall match the type of element of the corresponding rank in the union type description.

```
ConstantValue         ::=       BOOLEAN
                      /         OCTET
                      /         INTEGER       //     all numeric types
                      /         REAL              //     float or double
                      /         STRING            //     character or string
                      /         DataIdentifier
                      /         ConstantValue*    //     sequence, array or structure
                      /         UnionValue

UnionValue            ::=       INTEGER       //     Tag index
                                ConstantValue
```

12.3 Global variable declaration

Global variable declarations shall be used to describe the types and initial values of the global variables of the interchanged script.

A global variable declaration shall consist of:

- a data identifier (optional);
- a type identifier;
- a constant reference (optional).

```
GlobalVariableDeclaration   ::=    DataIdentifier?
                                   TypeIdentifier
                                   ConstantReference?   //  Initial value
```

12.3.1 Data identifier

DIDs are used to reference data throughout the interchanged script.

The DID shall be a positive integer within the range allowed for global variables. It shall correspond to the maximum number of constants incremented by the index (starting from 0) of the declaration in the global variable declarations part.

If the DID is not provided, it shall be computed by the script parser.

12.3.2 Type identifier

The TID represents the type to which the value of the global variable belongs.

12.3.3 Constant reference

The constant reference represents the initial value of the global variable.

The constant reference shall be either:

- a DID referencing a constant;
- a positive integer representing the index of the constant in the constant table;
- a constant value as described in subclause 12.2.3.

In any case, the value to which this constant reference refers shall be of a type that matches the type of the global variable.

If the constant reference is not provided, the global variable is assigned to a default value (if its type allows for it) or simply undefined until assigned by an instruction.

```
ConstantReference     ::=       DataIdentifier
                      /         INTEGER              //     constant index
                      /         ConstantValue
```

12.4 Package declaration

Package declarations shall be used to describe the external services and exceptions used by the interchanged script.

A package declaration shall consist of:

- a package identifier (optional);
- a string representing the package name;
- a sequence of service descriptions;
- a sequence of exception descriptions.

```
PackageDeclaration    ::=       PackageIdentifier?
                                VisibleString        //     Package name
                                ServiceDescription*
                                ExceptionDescription*
```

12.4.1 Package identifier

Package IDentifiers (PID) are used to reference packages throughout the interchanged script.

The PID shall be a positive integer within the range allowed for packages. It shall correspond to the index (starting from 0) of the declaration in the package declarations part.

If the PID is not provided, it shall be computed by the script parser.

```
PackageIdentifier     ::=       INTEGER
```

12.4.2 Package name

Package names allow the script interpreter to access the package within the run-time environment, according to the package availability procedure described by the platform mapping specification.

12.4.3 Service description

Service descriptions describe external function prototypes.

A service description shall consist of:

- a function identifier (optional);
- a string representing the operation name;
- a calling mode (optional);
- a type identifier;
- a sequence of parameter descriptions.

```
ServiceDescription   ::=       FunctionIdentifier?
                               VisibleString?       //     IDL global name
                               CallingMode?
                               TypeIdentifier       //     return value
                               ServiceParameterDescription*
```

12.4.3.1 Function identifier

Function IDentifiers (FIDs) are used to reference functions throughout the interchanged script.

The FID shall be a positive integer within the range allowed for services. It shall correspond to the maximum number of routines plus the maximum number of predefined functions plus the package identifier multiplied by 256, incremented by the index (starting from 0) of the service in the package declaration.

If the FID identifier is not provided, it shall be computed by the script parser.

```
FunctionIdentifier       ::=       INTEGER
```

12.4.3.2 Operation name

Operation names allow the script interpreter to access the operation within the run-time environment, according to the operation invocation procedure described by the platform mapping specification.

12.4.3.3 Calling mode

The calling mode represents the way the operation shall be invoked.

The calling mode shall be either "synchronous" or "asynchronous".

```
CallingMode          ::=       "SYNCHRONOUS"     /      "ASYNCHRONOUS"
```

12.4.3.4 Type identifier

The TID represents the type of return value of the service.

12.4.3.5 Parameter description

Parameter descriptions are used to specify the type and passing mode of service parameters.

A parameter description shall consist of:

- a passing mode;
- a type identifier.

```
ServiceParameterDescription     ::=    ServicePassingMode?
                                          TypeIdentifier
```

12.4.3.5.1 Passing mode

The passing mode indicates whether the value of the parameter at the time of invocation of the service shall be used by the service (input parameter) and whether this parameter shall be modified by the service for use by its caller (output parameter).

The passing mode shall be either "in", "inout" or "out".

```
ServicePassingMode              ::=    "IN" / "OUT" / "INOUT"
```

12.4.3.5.2 Type identifier

The TID represents the type of the considered service parameter.

12.4.4 Exception description

Exception descriptions describe prototypes of exceptions that may be raised during the execution of external functions.

An exception description shall consist of:

- a message identifier (optional);
- a string representing the exception name;
- a sequence of type identifiers representing the members of the exception.

```
ExceptionDescription ::=       MessageIdentifier?
                               VisibleString?    //IDL exception global name
                               TypeIdentifier*   //Parameter types
```

12.4.4.1 Message identifier

Message IDentifiers (MIDs) are used to reference messages throughout the interchanged script.

The MID shall be a positive integer within the range allowed for exceptions. It shall correspond to the maximum number of predefined messages plus the PID multiplied by 256, incremented by the index (starting from 0) of the exception in the package declaration.

If the MID is not provided, it shall be computed by the script parser.

```
MessageIdentifier    ::=       INTEGER
```

12.4.4.2 Exception name

Exception names allow the script interpreter to retrieve the exception within the run-time environment, according to the exception retrieval procedure described by the platform mapping specification.

12.4.4.3 Parameter description

Each parameter of the message corresponds to one member of the exception. It is described by its TID.

12.5 Handler declaration

Handler declarations shall be used to associate a message with the function that handles it.

A handler declaration shall consist of:

- a message reference;
- a function reference.

```
HandlerDeclaration    ::=       MessageReference
                                FunctionReference
```

12.5.1 Message reference

The message reference indicates the message to be handled.

A message reference shall be either:

- a message identifier;
- a string representing an exception name;
- an exception reference;
- an integer corresponding to the index of the message in the predefined messages table.

```
MessageReference        ::=     MessageIdentifier
                        /       VisibleString           //      exception name
                        /       ExceptionReference
                        /       INTEGER             //      predefined message index
```

12.5.1.1 Message identifier

The MID shall be a positive integer within the whole range allowed to messages, representing either a predefined message or an exception.

12.5.1.2 Exception reference

An exception reference shall consist of:

- a PID representing the package in which the exception is described;
- an integer representing the index of the exception description in the package.

```
ExceptionReference      ::=     PackageIdentifier
                                INTEGER          //  exception index
```

12.5.2 Function reference

The function reference indicates the function to be triggered when the message is removed from the message queue.

The description of the formal parameter types for the function shall be the same as for the message, so that the function may be called with the message actual parameters as its parameters. If parameters descriptions do not match, the handler shall not be invoked by the script parser.

A function reference shall be either:

- a function identifier;
- a service reference;
- an integer corresponding to the index of the routine in the routine declarations part;
- an integer corresponding to the index of the function in the predefined functions table.

```
FunctionReference     ::=       FunctionIdentifier
                      /         ServiceReference
                      /         INTEGER    //        routine index,
                                                //        predefined message index
```

12.5.2.1 Function identifier

The FID shall be a positive integer within the whole range allowed to function, representing either a routine, a predefined function or a service.

12.5.2.2 Service reference

An service reference shall consist of:

- a PID representing the package in which the service is described;
- an integer representing the index of the service description in the package.

```
ServiceReference     ::=       PackageIdentifier
                               INTEGER              //     service index
```

12.6 Routine declaration

Routine declarations shall be used to describe the structure and program code of the internal functions of the interchanged script.

A routine declaration shall consist of:

- a function identifier (optional);
- a type identifier;
- a sequence of parameter descriptions;
- a sequence of local variable declarations;
- MHEG-SIR program code.

```
RoutineDeclaration   ::=       FunctionIdentifier?
                               TypeIdentifier              // for return value
                               RoutineParameterDescription*
                               LocalVariableDeclaration*
                               OCTET STRING                   // program code
```

12.6.1 Function identifier

The FID shall be a positive integer within the range allowed for routines. It shall correspond to the index (starting from 0) of the routine in the routine declarations part.

If the FID is not provided, it shall be computed by the script parser.

12.6.2 Type identifier

The type identifier represents the type of return value of the routine.

12.6.3 Parameter description

Parameter descriptions are used to specify the type and passing mode of routine parameters.

A parameter description shall consist of:

- a passing mode (optional);
- a type identifier.

```
RoutineParameterDescription     ::=    RoutinePassingMode?
                                           TypeIdentifier
```

12.6.3.1 Passing mode

The passing mode indicates whether the parameter shall be passed to the routine using its value (input parameter) or a reference to the variable that holds its value (input/output parameter).

The passing mode shall be either "value" or "reference".

```
RoutinePassingMode          ::=    "VALUE" / "REFERENCE"
```

12.6.3.2 Type identifier

The TID represents the type of the considered routine parameter.

12.6.4 Local variable declaration

Local variable declarations shall be used to describe the types and initial values of variables whose scope is limited to one execution of a routine.

A local variable declaration shall consist of:

- a data identifier (optional);
- a type identifier;
- a constant reference (optional).

```
LocalVariableDeclaration        ::=    DataIdentifier?
                                       TypeIdentifier
                                       ConstantReference? // initial value (constant)
```

12.6.4.1 Data identifier

The DID shall be a positive integer within the range allowed for local variables. It shall correspond to the maximum number of constants plus the maximum number of global variables incremented by the index (starting from 0) of the declaration in the local variable declarations of the routine, incremented by the number of formal parameters of the routine.

If the DID is not provided, it shall be computed by the script parser.

12.6.5 Type identifier

The type identifier represents the type to which the value of the local variable belongs.

12.6.6 Constant reference

The constant reference represents the initial value of the local variable.

The constant reference shall be either

- a data identifier referencing a constant;
- a positive integer representing the index of the constant in the constant table;
- a constant value as described in subclause 12.2.3.

In any case, the value to which this constant reference refers shall be of a type that matches the type of the local variable.

If the constant reference is not provided, the local variable is assigned to a default value (if its type allows for it) or simply undefined until assigned by an instruction.

12.6.7 Program code

The program code consists of the sequence of instructions of the routine, intended for execution by the script interpreter when the routine is triggered. The syntax and semantics of the MHEG-SIR instructions are described in clause 14.

The last instruction of a routine shall always be a RET instruction.

13 MHEG-SIR instructions

This clause defines the semantics of the MHEG-SIR instructions.

13.1 Presentation methodology

Each instruction is described under the "instruction name" subclause by an entry of the following form:

Short description:	A brief description of the instruction's semantics.
Synopsis:	InstructionMnemonic Operand1 to OperandN.
Operands:	A description of the types and semantics of each operand carried with the instruction (if any).
Stack:	A visual synopsis of the instruction's effect on the parameter stack, e.g. ..., Parameter1, Parameter2 ⇨ to, Result.
Parameters:	A description of the types and semantics of each element of the parameter stack which is popped, pushed or otherwise effected by the instruction (if any). Type abbreviations are: B (boolean), C (character), D (double), F (float), L (long), O (octet), S (short), U (unsigned long), W (unsigned short).
Effect:	A textual specification of the interpretation semantics of the instruction.
Formal specification:	A formal specification of the interpretation semantics of the instruction using the notation described in subclause 13.2.
Errors:	A list of the (non-trivial) error cases that may occur during interpretation of the instruction.

The semantics of the instruction, as described by the formal specification, shall apply only if the following conditions are met:

- the operands are valid;
- the stack holds enough parameters;
- the types of the stack parameters are valid.

Otherwise an error shall be raised and the error register shall be set.

The formal specification part gives a concise formal notation of what effect that the instruction execution unit shall produce upon interpreting the instruction. However, since this specification is expressed in terms of a sequence of operations, there may be other methods to lead to the same result. This formal specification does not therefore require the instruction execution unit to perform such functionality as expressed, providing the effect is the same.

NOTE: The current description of the formal specification part does not describe all error cases.

13.2 Notation

To specify the semantics of an instruction in a formal way, a syntax similar to C is used. It uses the notations and concepts defined in clause 10, plus the following "macro" notations:

- variable table notation;
- data table notation;
- type matching notation.

13.2.1 Variable table notation

The notation VT(i), where i stands for the data identifier of a variable, corresponds to:

- the entry whose key is i in the global variable table, if i is the data identifier of a global variable;

- the entry whose key is i in the local variable table of the currently executing routine, if i is the data identifier of a local variable.

This macro may be expressed as follows:

```
#define VT(i) ( i < 32768) ? GT[i] : CS[FP].LT[i]
```

13.2.2 Data table notation

The notation DT(i), where i stands for a data identifier, corresponds to:

- the entry whose key is i in the constant table, if i is the data identifier of a constant;

- the entry whose key is i in the global variable table, if i is the data identifier of a global variable;

- the entry whose key is i in the local variable table of the currently executing routine, if i is the data identifier of a local variable.

This macro may be expressed as follows:

```
#define DT(i) ( i < 32768) ? ((i < 4096) ? CT[i] : GT[i]) : CS[FP].LT[i]
```

13.2.3 Type matching notation

The notations :-: is used to check whether two types match.

If TID1 and TID2 are two type identifiers:

- (TID1 :-: TID2) is true if and only if there is a formal matching between the two types (as defined in clause 9).

13.2.4 Type combination

All arithmetic and logical instructions operate on values of a given type and produce a result with the same type. Operations on mixed types shall be handled by explicitly inserting type conversion instructions in the instruction sequence.

13.3 Classification of MHEG-SIR instructions

The MHEG-SIR instructions may be clustered into categories according to their effect on the control flow, on the variable tables or on the parameter stack, and according to the types of stack parameters that they accept:

- instructions which affect the control flow:

  a) unconditional jump instructions: JMP, LJMP;
  b) conditional jump instructions: JT, JF, LJT, LJF;
  c) function calls: CALL, XCALL;
  d) miscellaneous control flow instructions: RET, YIELD.

- instructions which do not affect the control flow, but affect the value of variables:

  a) complex variable modifiers: SET, SETC;
  b) arithmetic operators on variables: INC, DEC;
  c) stack pop instructions: POPR, POP, POPC.

- instructions which do not affect the control flow or the global variables, but affect the parameter stack:

  a) arithmetic operators: ADD, SUB, MUL, DIV, REM, NEG;
  b) logical operators: AND, OR, XOR, NOT;
  c) logical shift operators: SHIFT;
  d) comparison operators: EQ, NEQ, GT, GE, LT, LE;
  e) complex data accessors: GET;
  f) miscellaneous stack manipulation instructions: PUSHI, PUSHR, PUSH, DROP; DUP.

- memory management instructions: ALLOC, FREE;

- instructions which have no effect: NOP.

  NOTE: The use of conditional flags such as arithmetic carry, overflow, zero is to be studied. If used, flags would be set by the result of arithmetic and logical instructions, and would be used in conditional flow of control instructions.

The effect of instructions is summarised in table 2. The operations are listed in canonical order, i.e. by ascending op-code number. Some mnemonics have type suffixes. The suffix "x" can be one of: {O, S, W, L, U, F, D}. The suffix "y" can be one of {O, W, U, F, D, B}. The suffix "z", "z1", and "z2" can be one of: {O, C, S, W, L, U, F, D}. For instructions with type suffixes, opcodes are assigned in ascending order in the range indicated.

Table 2: Synopsis of MHEG-SIR instructions and their effect

Mnemonics	Code(s)	Op. size	Op. type	PS effect	VT/LT effect	Control flow effect
NOP	00h	0				
YIELD	01h	0				x
RET	02h	0		0/1 ⇨ 0/1		x
ADDx	03h - 09h	0		2 ⇨ 1		
SUBx	0Ah - 10h	0		2 ⇨ 1		
MULx	11h - 17h	0		2 ⇨ 1		
DIVx	18h - 1Eh	0		2 ⇨ 1		
REMx	1Fh - 25h	0		2 ⇨ 1		
NEGx	26h - 2Ch	0		1 ⇨ 1		
NOTy	2Dh - 32h	0		1 ⇨ 1		
ANDy	33h - 38h	0		2 ⇨ 1		
ORy	39h - 3Eh	0		2 ⇨ 1		
XORy	3Fh -44h	0		2 ⇨ 1		
EQz	45h - 4Ch	0		2 ⇨ 1		
LEx	4Dh - 53h	0		2 ⇨ 1		
GTx	54h - 5Ah	0		2 ⇨ 1		
JT	5Bh	1	offset	1 ⇨ 0		x
JF	5Ch	1	offset	1 ⇨ 0		x
JMP	5Dh	1	offset			x
DROP	5Eh	1	idx	idx ⇨ 0		
SHIFTy	5Fh - 64h	1	offset	1 ⇨ 0		
LJT	65h	2	offset	1 ⇨ 0		x
LJF	66h	2	offset	1 ⇨ 0		x
LJMP	67h	2	offset			x
CALL	68h	2	FID	n ⇨ 0/1		x
XCALL	69h	2	FID	n ⇨ 0/1		x
PUSHI	6Ah	2	value	0 ⇨ 1		
PUSH	6Bh	2	DID	0 ⇨ 1		
PUSHR	6Ch	2	DID	0 ⇨ 1		
POP	6Dh	2	DID	1 ⇨ 0	x	
POPR	6Eh	2	DID	1 ⇨ 0	x	
POPC	6Fh	2	DID	1 ⇨ 0	x	
INC	70h	2	DID	1 ⇨ 0	x	
DEC	71h	2	DID	1 ⇨ 0	x	
GET	72h	3	DID, idx	idx ⇨ 1		
SET	73h	3	DID, idx	idx+1 ⇨ 0	x	
SETC	74h	3	DID, idx	idx+1 ⇨ 0	x	
ALLOC	75h	2	TID	0 ⇨ 1	x	
FREE	76h	0		1 ⇨ 0	x	
DUPz	77h - 7Dh	0		1 ⇨ 2		
CVTz1z2	80h - ACh	0		1 ⇨ 1		

13.4 Description of instructions

13.4.1 No operation

Short description: Does nothing.

Synopsis: NOP.

Operands: None.

Parameters: None.

Stack: ... ⇨ ...

Effect: None.

Formal specification: 0;

Errors:

13.4.2 Yield

Short description: Yield control.

Synopsis: YIELD.

Operands: None.

Stack: ... ⇨ ...

Parameters: None.

Effect: Yield control to the MHEG engine.

NOTE 1: As control returns, the script interpreter will first look up at the message queue and possibly stack one calling frame to handle its messages. Once the CS is restored to the same state as before the instruction, the next instruction will be executed.

NOTE 2: Relevance of introducing a synchronization mechanism similar to a WAIT TID instruction needs further study.

NOTE 3: How concurrency between the script interpreter and the MHEG engine is handled is a system design issue.

Formal specification: 0;

Errors: None.

13.4.3 Return

Short description: Return to caller.

Synopsis: RET.

Operands: None.

Stack: ..., (Val) ⇨ ..., (Val).

Parameters: If the current routine signature has a return value, Val shall be of a type matching the type of this return value.
Otherwise, there is no stack parameter.

Effect: Return to the calling routine. Pop the calling stack and restore the context of the previous frame. If the current routine has a return value, there shall be a value of a type matching this type on the top of the parameter stack.
If there is no calling function to return to, yield control to the MHEG engine.

Formal specification:
```
if ((RT[CS[FP].FID].TID :-: "void") // (RT[CS[FP].FID].TID :-: PS[SP].TID))
        IP = CS[FP--].IP;
else ER = "invalid return value";
```

Errors: Invalid return value.

13.4.4 Add

Short description: Arithmetic addition.

Synopsis: ADD<type>.

Operands: None.

Stack: ..., Num1, Num2 ⇨ ..., Sum.

Parameters: Num1 and Num2 shall be of the same numeric type, where type is one of {O, S, W, L, U, F, D}.
Sum shall have the corresponding type.

Effect: Replace the top two elements of the parameter stack by their sum:
Sum = Num1 + Num2.

Formal specification: PS[SP-1].val += PS[SP].val;
SP -= sizeof(<type>);

Errors: Void stack;
Invalid parameter type;
Result out of range.

13.4.5 Subtract

Short description: Arithmetic subtraction.

Synopsis: SUB<type>.

Operands: None.

Stack: ..., Num1, Num2 ⇨ ..., Diff.

Parameters: Num1 and Num2 shall be of the same numeric type, where type is one of {O, S, W, L, U, F, D}.
Difference shall have the corresponding type.

Effect: Replace the top two elements of the parameter stack by their difference:
Diff = Num1 - Num2.

Formal specification: PS[SP-1].val -= PS[SP].val;
SP -= sizeof(<type>);

Errors: Void stack;
Invalid parameter type;
Result out of range.

13.4.6 Multiply

Short description: Arithmetic multiplication.

Synopsis: MUL<type>.

Operands: None.

Stack: ..., Num1, Num2 ⇨ ..., Prod.

Parameters: Num1 and Num2 shall be of the same numeric type, where type is one of {O, S, W, L, U, F, D}
Product shall have the corresponding type.

Effect: Replace the top two elements of the parameter stack by their product:
Prod = Num1 * Num2.

Formal specification: PS[SP-1].val *= PS[SP].val;
SP -= sizeof(<type>);

Errors: Void stack;
Invalid parameter type;
Result out of range.

13.4.7 Divide

Short description: Arithmetic division.

Synopsis: DIV<type>.

Operands: None.

Stack: ..., Num1, Num2 ⇨ ..., Quot.

Parameters: Num1 and Num2 shall be of the same numeric type, where type is one of {O, S, W, L, U, F, D}.
Quotient shall have the corresponding type.

Effect: Replace the top two elements of the parameter stack by their quotient:
Prod = Num1 / Num2.

Formal specification: PS[SP-1].val /= PS[SP].val;
SP -= sizeof(<type>).

Errors: Void stack;
Invalid parameter type;
Division by 0.

13.4.8 Remainder

Short description: Arithmetic remainder.

Synopsis: REM<type>.

Operands: None.

Stack: ..., Num1, Num2 ⇨ ..., Rem.

Parameters: Num1 and Num2 shall be of the same numeric type, where type is one of {O, S, W, L, U, F, D}.
Remainder shall have the corresponding type.

Effect: Replace the top two elements of the parameter stack by their remainder:
Prod = Num1 % Num2.

Formal specification: PS[SP-1].val %= PS[SP].val;
SP -= sizeof(<type>);

Errors: Void stack;
Invalid parameter type;
Division by 0.

13.4.9 Negate

Short description: Sign change.

Synopsis: NEG<type>.

Operands: None.

Stack: ..., Num ⇨ ..., Opp.

Parameters: Num shall be one of { O, S, W, L, U, F, D}.
Result shall have the corresponding type.

Effect: Replace the top element of the parameter stack by its opposite (for a signed type) or its complement-to-two (for an unsigned type):
Opp = -Num1.

Formal specification: PS[SP].val = -PS[SP].val;

Errors: Void stack;
Invalid parameter type.

13.4.10 Not

Short description: Logical negation.

Synopsis: NOT<type>.

Operands: None.

Stack: ..., Val ⇨ ..., Neg.

Parameters: Val shall be of type: { O, W, U, F, D, B }.
Neg shall be of the same type as Val.

Effect: Replace the top element of the parameter stack by its logical negation (for a boolean) or its bitwise negation (i.e. complement-to-one, for an octet or integer type):
Neg = ~Val.

Formal specification:
```
if (PS[SP].TID :-: "boolean")
   PS[SP].val = ! PS[SP].val;
else
   PS[SP].val = ~ PS[SP].val;
```

Errors: Void stack;
Invalid parameter type.

13.4.11 And

Short description: Logical conjunction.

Synopsis: AND<type>.

Operands: None.

Stack: ..., Val1, Val2 ⇨ ..., Conj.

Parameters: Val1 and Val2 shall be of the same type, which is one of: {O, W, U, F, D, B }. The result is of the same type.

Effect: Replace the top two elements of the parameter stack by their logical conjunction (for a boolean) or their bitwise conjunction (for an octet or integer type): Conj = Val1 & Val2.

Formal specification:
```
if (PS[SP].TID :-: "boolean")
  PS[SP-1].val = PS[SP].val && PS[SP-1].val;
else
  PS[SP-1].val &= PS[SP].val;
SP -= sizeof(<type>);
```

Errors: Void stack;
Invalid parameter type;
Parameter type mismatch.

13.4.12 Or

Short description: Logical disjunction.

Synopsis: OR.

Operands: None.

Stack: ..., Val1, Val2 ⇨ ..., Disj.

Parameters: Val1 and Val2 shall be of the same type, which is one of: {O, W, U, F, D, B }. The result is of the same type.

Effect: Replace the top two elements of the parameter stack by their logical disjunction (for a boolean) or their bitwise disjunction (for an octet or integer type): Disj = Val1 / Val2.

Formal specification:
```
if (PS[SP].TID :-: "boolean")
  PS[SP-1].val = PS[SP].val // PS[SP].val;
else
  PS[SP-1].val /= PS[SP].val;
SP -= sizeof(<type>);
```

Errors: Void stack;
Invalid parameter type;
Parameter type mismatch.

13.4.13 Exclusive or

Short description: Logical exclusion.

Synopsis: XOR<type>.

Operands: None.

Stack: ..., Val1, Val2 ⇨ ..., Excl.

Parameters: Val1 and Val2 shall be of the same type, which is one of: {O, W, U, F, D, B }. The result is of the same type.

Effect: Replace the top two elements of the parameter stack by their logical exclusion (for a boolean) or their bitwise exclusion (for an octet or integer type):
Excl = Val1 ^ Val2.

Formal specification:
```
if (PS[SP].TID :-: "boolean")
  PS[SP-1].val = (PS[SP-1].val != PS[SP].val);
else
  PS[SP-1].val ^= PS[SP].val;
SP -= sizeof(<type>);
```

Errors: Void stack;
Invalid parameter type;
Parameter type mismatch.

13.4.14 Equal

Short description: Equality.

Synopsis: EQ<type>.

Operands: None.

Stack: ..., Val1, Val2 ⇨ ..., Comp.

Parameters: Val1 and Val2 shall be of the same type, which is one of:
{O, C, S, W, L, U, F, D }. The result is of type boolean.

Effect: Replace the top two elements of the parameter stack by "true" if they are equal and "false" otherwise:
Comp = (Val1 == Val2).

Formal specification:
```
PS[SP-1].val = (PS[SP-1].val == PS[SP].val);
SP = SP - 2*sizeof(<type>) + sizeof(boolean);
```

Errors: Void stack;
Invalid parameter type;
Parameter type mismatch.

13.4.15 Less or equal

Short description: Inferiority.

Synopsis: LE<type>.

Operands: None.

Stack: ..., Val1, Val2 ⇨ ..., Comp.

Parameters: Val1 and Val2 shall be of the same type, which is one of { O, S, W, L., U, F, D }.
Comp shall be of boolean type.

Effect: Replace the top two elements of the parameter stack by "true" if the top element is greater than the next or if they are equal, and "false" otherwise:
Comp = (Val1 <= Val2).
If parameters are characters, the canonical order is used.

Formal specification:
```
PS[SP-1].val = (PS[SP-1].val <= PS[SP].val);
SP = SP - 2*sizeof(<type>) + sizeof(boolean);
```

Errors: Void stack;
Invalid parameter type;
Parameter type mismatch.

13.4.16 Greater than

Short description: Strict superiority.

Synopsis: GT<type>.

Operands: None.

Stack: ..., Val1, Val2 ⇨ ..., Comp.

Parameters: Val1 and Val2 shall be of the same type, which is one of { O, S, W, L., U, F, D }.
Comp shall be of boolean type.

Effect: Replace the top two elements of the parameter stack by "true" if the top element is less than the next, and "false" otherwise:
Comp = (Val1 > Val2).
If parameters are characters, the canonical order is used.

Formal specification: PS[SP-1].val = (PS[SP-1].val > PS[SP].val);
SP = SP - 2*sizeof(<type>) + sizeof(boolean);

Errors: Void stack;
Invalid parameter type;
Parameter type mismatch.

13.4.17 Jump on true

Short description: "If" conditional short jump.

Synopsis: JT Off.

Operands: Off shall be a one-byte signed offset (in complement-to-two notation) specifying the number of instructions to move forwards or backwards within the current routine.

Stack: ..., Test ⇨ ...

Parameters: Test shall be of a boolean, octet or integer type.

Effect: If the top element of the stack is "true" or different from 0:
if Off is positive, jump Off instructions forwards;
if Off is negative, jump -Off instructions backwards.

Formal specification: if (PS[SP--].val) then IP += Off;

Errors: Void stack;
Invalid parameter type;
Jump out of range.

13.4.18 Jump on false

Short description: "Else" conditional short jump.

Synopsis: JF Off.

Operands: Off shall be a one-byte signed offset (in complement-to-two notation) specifying the number of instructions to move forwards or backwards within the current routine.

Stack: ..., Test ⇨ ...

Parameters: Test shall be of a boolean, octet or integer type.

Effect: If the top element of the stack is "false" or 0:
if Off is positive, jump Off instructions forwards;
if Off is negative, jump -Off instructions backwards.

Formal specification: if !(PS[SP--].val) then IP += Off;

Errors: Void stack;
Invalid parameter type;
Jump out of range.

13.4.19 Jump

Short description: Unconditional short jump.

Synopsis: JMP Off.

Operands: Off shall be a one-byte signed offset (in complement-to-two notation) specifying the number of instructions to move forwards or backwards within the current routine.

Stack: ... ⇨ ...

Parameters: None.

Effect: If Off is positive, jump Off instructions forwards;
if Off is negative, jump -Off instructions backwards.

Formal specification: IP += Off;

Errors: Jump out of range.

13.4.20 Long jump on true

Short description: "If" conditional long jump.

Synopsis: LJT Off.

Operands: Off shall be a two-byte signed offset (in complement-to-two notation) specifying the number of instructions to move forwards or backwards within the current routine.

Stack: ..., Test ⇨ ...

Parameters: Test shall be of a boolean, octet or integer type.

Effect: If the top element of the stack is "true" or different from 0:
if Off is positive, jump Off instructions forwards;
if Off is negative, jump -Off instructions backwards.

Formal specification: if (PS[SP--].val) then IP += Off;

Errors: Void stack;
Invalid parameter type;
Jump out of range.

13.4.21 Long jump on false

Short description: "Else" conditional long jump.

Synopsis: LJF Off.

Operands: Off shall be a two-byte signed offset (in complement-to-two notation) specifying the number of instructions to move forwards or backwards within the current routine.

Stack: ..., Test ⇨ ...

Parameters: Test shall be of a boolean, octet or integer type.

Effect: If the top element of the stack is "false" or 0:
if Off is positive, jump Off instructions forwards;
if Off is negative, jump -Off instructions backwards.

Formal specification: if !(PS[SP--].val) then IP += Off;

Errors: Void stack;
Invalid parameter type;
Jump out of range.

13.4.22 Long jump

Short description: Unconditional long jump.

Synopsis: LJMP Off.

Operands: Off shall be a two-byte signed offset (in complement-to-two notation) specifying the number of instructions to move forwards or backwards within the current routine.

Stack: ... ⇨ ...

Parameters: None.

Effect: If Off is positive, jump Off instructions forwards;
if Off is negative, jump -Off instructions backwards.

Formal specification: IP += Off;

Errors: Jump out of range.

13.4.23 Call

Short description: Call routine.

Synopsis: CALL Fid.

Operands: Fid specifies the function identifier of the routine to invoke.

Stack: ..., ParN, ... , Par1 ⇨ ...

Parameters: Par1, ..., ParN are the actual parameters of the routine. They shall be of the same type as the formal parameters of the function when those are passed by value, and they shall be of "data identifier" type and reference a variable of the same type of the formal parameters of the routine while those are passed by reference.

Effect: Pop the top elements of the parameter stack and invoke the routine specified by Fid with these elements as actual parameters. Push one frame onto the calling stack with the current context. Initialises the local variable table for the routine. Set the instruction pointer to the first instruction of the routine.

Formal specification:

```
CS[++FP].IP = IP;
CS[FP].SP = SP;
CS[FP].FID = Fid;
CS[FP].LT = RT[Fid].LT;
for (short i = 0; i<RT[Fid].nbp; i--;)       {
        switch (RT[Fid].sig[i].mod) {
        case "value":
                if !(RT[Fid].sig[i].TID :-: PS[SP].TID)
                then exit (ER = "type mismatch");
        case "reference":
                if !((PS[SP].TID == "data identifier") &&
(RT[Fid].sig[i].TID :-: VT(PS[SP].val).TID))
                then exit (ER = "type mismatch");
        };
        CS[FP].LT[i+0x8000].val = PS[SP--].val;
};
IP = RT[Fid].IP;
```

Errors: Invalid function identifier;
Void stack;
Invalid parameter type;
Type mismatch;
Invalid return value.

13.4.24 External call

Short description: Call external function.

Synopsis: XCALL Fid.

Operands: Fid specifies the function identifier of the service or MHEG operation to invoke.

Stack: ..., ParN, ... , Par1 ⇨ ..., (Ret).

Parameters: Par1, ..., ParN are the actual parameters of the function. Whatever the passing mode, they shall be of "data identifier" type and reference a variable of the same type of the formal parameters of the function.
If the function has a return value type other than void, Ret shall be of this type.

Effect: Pop the top elements of the parameter stack and invoke the external function specified by Fid with these elements as actual parameters. Push one frame onto the calling stack with the current context. Pass parameters to and invoke the external function. If the invocation is synchronous, handle any exceptions that may be raised by it. Otherwise retrieve its return value and push it onto the parameter stack. Pop the calling stack.

Formal specification:

```
CS[++FP].IP = IP;
CS[FP].SP = SP;
CS[FP].FID = Fid;
CS[FP].LT = NULL;
DID buf[ST[Fid].nbp];
for (short i=0; i<ST[Fid].nbp; i++;)        {
        if !((PS[SP].TID == "data identifier") &&
(ST[Fid].sig[i].TID :-: VT(PS[SP].val).TID))
                then exit (ER = "type mismatch");
        buf[i]=PS[SP--].val;
};
short Pid = (Fid>>8)-32;
if (PT[Pid].sts == "available")  _package_load_procedure(PT[Pid].name);
for (short i=0;         i<ST[Fid].nbp; i++;)
        switch(ST[Fid].sig[i].mod){
        case "in": _in_parameter_passing _procedure(buf[i]);
        case "out": _inout_parameter_passing _procedure(buf[i]);
        case "inout": _out_parameter_passing _procedure(buf[i]);
        };
_operation_invocation_procedure(PT[Pid].name, ST[Fid].name);
// this involves yielding control;
// stacks and registers may be modified in the meanwhile;
// the current description does not deal with exceptions raised by the call;
_output_parameter_retrieval_procedure();
SP = CS[FP].SP - ST[Fid].nbp;
if (ST[Fid].TID != "void") PS[++SP] = _return_value _retrieval_procedure();
IP = CS[FP--].IP;
```

Errors:

Invalid function identifier;
Void stack;
Invalid parameter type;
Type mismatch;
Package not accessible;
Operation not available.

13.4.25 Drop

Short description: Multiple pop.

Synopsis: DROP Idx.

Operands: Idx shall be a one-byte unsigned quantity specifying the number of places to move downwards in the parameter stack.

Stack: ..., Val(1),...,Val(Idx) ⇨ ...

Parameters: Val(1), ... Val(Idx) may be of any type.

Effect: Drop Idx elements from the parameter stack.

Formal specification: SP -= Idx.

Errors: Void stack.

13.4.26 Shift

Short description: Logical shift.

Synopsis: SHIFT<type> Off.

Operands: Off shall be a one-byte signed offset (in complement-to-two notation) specifying the number of bit places to shift the parameter leftwards or rightwards.

Stack: ..., Val ⇨ ..., Pwr.

Parameters: Val shall be of type { O, W, U }.
Pwr shall be of the same type as Val.

Effect: Replace the top element of the stack by its value shifted right Off bits if Off is positive, or left -Off bits if Off is negative. In the latter case, if Val1 is a signed integer, preserve the sign (i.e. do not affect the msb):

Formal specification:
```
if (Off >=0) then PS[SP].val >>= Off;
else if (PS[SP].val < 0)
        PS[SP].val = -((-PS[SP].val) << -Off);
else PS[SP].val <<= -Off;
```

Errors: Void stack;
Invalid parameter type;
Shift out of range.

13.4.27 Push immediate

Short description: Push short integer.

Synopsis: PUSHI Int.

Operands: Int shall be the two-byte representation of a signed short integer value (in complement-to-two notation) specifying the value to push onto the stack.

Stack: ... ⇨ ..., Val.

Parameters: Val shall be of "short" type.

Effect: Push Int onto the parameter stack.

Formal specification:
```
PS[SP].val = Int;
SP += sizeof(short);
```
Errors: None.

13.4.28 Push

Short description: Push data value.

Synopsis: PUSH Did.

Operands: Did shall be the two-byte representation of a data identifier holding the value to push onto the stack.

Stack: ... ⇨ ..., Val.

Parameters: Val shall be of the same type as the constant or variable identified by Did.

Effect: Push the value of the constant or variable whose data identifier is Did onto the parameter stack.

Formal specification:
```
PS[SP].val = DT(Did).val;
SP += sizeof(DT(Did).TID);
```
Errors: Invalid data identifier.

13.4.29 Push reference

Short description: Push data identifier.

Synopsis: PUSHR Did.

Operands: Did shall be the two-byte representation of a data identifier to push onto the stack.

Stack: ... ⇨ ..., Val.

Parameters: Val shall be of "data identifier" type.

Effect: Push Did onto the parameter stack.

Formal specification:
```
SP += sizeof(VT(Did).TID);
PS[SP].val = Did;
```

Errors: None.

13.4.30 Pop

Short description: Pop value and assign it to a variable.

Synopsis: POP Did.

Operands: Did shall be the two-byte representation of the data identifier of the variable to which to assign the top element of the stack.

Stack: ..., Val ⇨ ...

Parameters: Val shall be of a type which can be converted to the type of the variable identified by Did.

Effect: Pop Val from the parameter stack into the variable identified by Did.

Formal specification:
```
VT(Did).val = PS[SP].val;
SP -= sizeof(VT(Did).TID);
```

Errors:
Invalid data identifier;
Void stack;
Type mismatch.

13.4.31 Pop reference

Short description: Pop value and assign it to the variable referenced by a variable.

Synopsis: POPR Did.

Operands: Did shall be the two-byte representation of the data identifier of a variable of "data identifier" type, whose value identifies the variable to which to assign the value of the top element of the stack. The variable identified by Did shall be of "data identifier" type.

Stack: ..., Val ⇨ ...

Parameters: Val shall be of a type that can be converted to the type of VT(Did).val.

Effect: Pop Val from the parameter stack and assigns it to the variable identified by Did.

Formal specification:
```
VT(VT(Did).val).val = PS[SP].val;
SP -= sizeof(VT(Did).TID);
```

Errors: Invalid data identifier;
Void stack;
Type mismatch.

13.4.32 Pop contents

Short description: Pop variable and assign its value to a variable.

Synopsis: POPC Did1.

Operands: Did1 shall be the two-byte representation of the data identifier of the variable to which to assign value of the data identified by the top element of the stack.

Stack: ..., Did2 ⇨ ...

Parameters: Did2 shall be of "data identifier" type, it shall identify a data whose type can be converted to the type of Did1.

Effect: Pop Did2 from the parameter stack and assigns the value of Did2 to the variable identified by Did1.

Formal specification: VT(Did1).val = DT(PS[SP].val).val;
SP -= sizeof(VT(Did).TID);
Errors: Invalid data identifier;
Void stack;
Type mismatch.

13.4.33 Increment

Short description: Increment variable.

Synopsis: INC Did.

Operands: Did shall be the two-byte representation of the data identifier of the variable which to increment. The variable identified by Did shall be of a numeric type.

Stack: ..., Val ⇨ ...,

Parameters: Val shall be of a numeric type that can be converted to the type of the variable identified by Did.

Effect: Pop the parameter stack and increment the value of the variable identified by Did by the popped value.

Formal specification: VT(Did).val += PS[SP].val;
SP -= sizeof(VT(Did).TID);
Errors: Invalid data identifier;
Void stack;
Type mismatch.

13.4.34 Decrement

Short description: Decrement variable.

Synopsis: DEC Did.

Operands: Did shall be the two-byte representation of the data identifier of the variable which to decrement. The variable identified by Did shall be of a numeric type.

Stack: ..., Val ⇨ ...,

Parameters: Val shall be of a numeric type that can be converted to the type of the variable identified by Did.

Effect: Pop the parameter stack and decrement the value of the variable identified by Did by the popped value.

Formal specification:
```
VT(Did).val -= PS[SP].val;
SP -= sizeof(VT(Did).TID);
```
Errors: Invalid data identifier;
Void stack;
Type mismatch.

13.4.35 Get

Short description: Get value of element of variable of constructed type.

Synopsis: GET Did Lvl.

Operands: Did shall be the two-byte representation of the data identifier of the variable to access.
Lvl shall be a one-byte unsigned quantity representing the number of nested levels to go to access the sought value.

Stack: ..., Idx(Lvl), ..., Idx(1) ⇨ ..., Val.

Parameters: Idx(1), ... Idx(Lvl) shall be of an "octet" or integer type; they shall represent unsigned positive values.
Val shall be of the same type as the accessed element.

Effect: Replace a list of indices on the parameter stack by the value of the element addressed by the popped indices within the structured variable identified by Did:

Val = DT(Did)[Idx(1),...,Idx(Lvl)].
If Lvl equals 0, perform as a PUSH instruction.

Formal specification:
```
void *buf = DT(Did);
int n = sizeof(VT(Did).TID);
for (;Lvl>0; Lvl--;) {
    buf = buf.val[PS[SP= n].val];
    SP -= n
}
PS[SP].val = buf.val;
```
Errors: Invalid data identifier;
Level out of range;
Void stack;
Invalid parameter type;
Invalid index.

13.4.36 Set

Short description: Set element of variable of constructed type to value.

Synopsis: SET Did Lvl.

Operands: Did shall be the two-byte representation of the data identifier of the variable to modify.
Lvl shall be a one-byte unsigned quantity representing the number of nested levels to go to access the element to modify.

Stack: ..., Val, Idx(Lvl), ..., Idx(1) ⇨ ...,

Parameters: Idx(1), ... Idx(Lvl) shall be of an "octet" or integer type; they shall represent unsigned positive values.
Val shall be of a type that matches the type of the element to modify.

Effect: Pop a list of indices and a value from the parameter stack; within the structured variable identified by Did, assign the element addressed by the popped list of indices to the popped value:
VT(Did)[Idx(1),...,Idx(Lvl)] = Val.
If Lvl equals 0, perform as a POP instruction.

Formal specification:
```
void *buf = VT(Did);
int n = sizeof(VT(Did).TID);
for (;Lvl>0; Lvl--;)   {
        if (buf.TID :--: PS[SP].TID) then {
                buf = buf.val[PS[SP].val];
                 SP -= n;
         }
        else exit (ER = "type mismatch");

};
```

Errors: Invalid data identifier;
Level out of range;
Void stack;
Invalid parameter type;
Index out of range;
Type mismatch.

13.4.37 Set contents

Short description: Set element of variable of constructed type to data contents.

Synopsis: SETC Did1 Lvl.

Operands: Did1 shall be the two-byte representation of the data identifier of the variable to modify.
Lvl shall be a one-byte unsigned quantity representing the number of nested levels to go to access the element to modify.

Stack: ..., Did2, Idx(Lvl), ..., Idx(1) ⇨ ...,

Parameters: Idx(1), ... Idx(Lvl) shall be of an "octet" or integer type; they shall represent unsigned positive values.
Did2 shall be of "data identifier" type and the type of the identified data shall match the type of the element to modify.

Effect: Pop a list of indices and a data identifier from the parameter stack; within the structured variable identified by Did1, assign the element addressed by the popped list of indices to the value identified by the popped data:
VT(Did1)[Idx(1),...,Idx(Lvl)] = DT(Did2).
If Lvl equals 0, perform as a POPC instruction.

Formal specification:
```
void *buf = VT(Did1);
for (;Lvl>0; Lvl--;)   {
        if (buf.TID :--: PS[SP].TID) then buf = buf.val[PS[SP--].val];
        else exit (ER = "type mismatch");
      SP -= sizeof(VT(Did).TID);
};
```

Errors: Invalid data identifier;
Level out of range;
Void stack;
Invalid parameter type;
Index out of range;
Type mismatch.

13.4.38 Alloc

Short description: Allocate data from dynamic memory and return a DID on the stack.

Synopsis: ALLOC TID.

Operands: TID is a legal type identifier appearing in the TID table.

Stack: ..., ⇨ DID, ...

Parameters: None.

Effect: Generates a new DID and associates a memory element of size sizeof(TID) with the DID. The two-byte DID is returned on the stack.

Formal specification:

Errors: Out of memory;
Invalid operand;
Invalid stack parameter.

13.4.39 Free

Short description: Release data previously allocated from dynamic memory.

Synopsis: FREE.

Operands: None.

Stack: ..., DID ⇨ ...

Parameters: DID shall be the two-byte representation of a data identifier which has been previously allocated using ALLOC.

Effect: The dynamic memory associated with the DID is released. The DID is now invalid in further operations.

Formal specification:

Errors: Invalid operand;
Invalid stack parameter.

13.4.40 Dup

Short description: Duplicate the top of stack element.

Synopsis: DUP<type>.

Operands: None.

Stack: ..., Val ⇨ Val, Val, ...

Parameters: None.

Effect: The value on the top of stack is duplicated according to the type of DUP, which is one of: {O, C, S, W, L, U, F, D}.

Formal specification: SP += sizeof(<type>)
PS[SP] = PS[SP-sizeof(<type>)"

Errors: Invalid operand;
Invalid stack parameter.

13.4.41 CVT

Short description: Convert the top of stack element from one primitive type to another.

Synopsis: CVT<src-type><dst-type>.

Operands: None.

Stack: ..., Val ⇨ ..., Val.

Parameters: None.

Effect: The value on the top of stack is replaced by an equivalent value in the destination type. Source and destination types are one of: {O, C, S, W, L, U, F, D}. When conversion leads to loss of precision, the source value is truncated to fit in the precision of the destination type.

Formal specification: SP = SP - sizeof(<src-type>) + sizeof(<dst-type>);

Errors: Invalid operand;
Invalid stack parameter.

14 IDL mapping to MHEG-SIR

This clause shows how an IDL specification shall be mapped to the declarations of an interchanged script, when this IDL specification is intended for use by the script as an external service provider.

This clause defines the mapping to MHEG-SIR declarations for:

- IDL interfaces and modules;
- IDL types;
- IDL constants;
- references to IDL objects;
- IDL operations;
- IDL attributes;
- IDL exceptions.

14.1 IDL specifications

An IDL specification shall be mapped to an MHEG-SIR `PackageDeclaration`. The name of the IDL specification shall be mapped to the `PackageName` field of this package declaration.

NOTE: Examples of IDL specifications are MHEG-API, MPEG/DSM-CC.

If the number of operations or exceptions of an IDL specification exceed the size of a package, it shall be divided between several packages sharing the same name, but having different MHEG-SIR identifiers.

14.2 IDL interfaces and modules

As the package declaration is a "flat" organisation, there is neither a mapping for an IDL module nor for an IDL interface. However, a reference to the embedding interface (i.e. a parameter of type Object) shall be provided as an implicit parameter to each invocation of function describing an IDL operation.

14.3 IDL operations

An IDL operation shall be mapped to an MHEG-SIR `ServiceDescription` within a package declaration that corresponds to the IDL specification it belongs to.

14.3.1 Operation name

The global name for an IDL operation shall be mapped to the MHEG-SIR `OperationName` field of this service description.

14.3.2 Operation parameters

The parameters of an IDL operation shall be mapped to the `ServiceParametersDescription` field of the service description. In this parameters description, each IDL parameter type shall be mapped to the `ParameterType` field which identifies a type declared according to the type mapping rules defined in this clause. The IDL passing mode for a parameter shall be mapped to the `PassingMode` field of the corresponding MHEG-SIR parameter description.

14.3.3 Implicit parameter

When an IDL operation is mapped to an MHEG-SIR service description, the signature of the service shall accept one additional leading parameter whose `ParameterType` shall be "object reference" and whose `PassingMode` shall be "in". This parameter represents a handle to the object instance to which the operation applies.

14.3.4 Return value

The return value type of an IDL operation shall be mapped to the `ReturnValueType` field of the service description.

14.4 IDL attributes

An IDL attribute shall be mapped to two service descriptions within a package declaration: one accessor service, whose function is to get the value of the attribute, and one modifier service, whose function is to set the value of the attribute.

14.4.1 Accessor

Concerning the accessor service, the global IDL attribute name postfixed with "_get” shall be mapped to the MHEG-SIR `OperationName`. An accessor service shall have no explicit parameter. The IDL attribute type shall be mapped to the ReturnValueType field of the service description.

14.4.2 Modifier

Concerning the modifier service, the global IDL attribute name postfixed with "_set” shall be mapped to the MHEG-SIR `OperationName`. A modifier service shall have one parameter with "in" passing mode and such that the IDL attribute type shall be mapped to the `ParameterType` field of the parameters description for this service. A modifier service shall have no return value.

14.4.3 Readonly attribute

If an IDL attribute is defined as "readonly", only the accessor service shall be provided as part of the package declaration.

14.5 IDL inherited operations

Inherited IDL operations shall be mapped as if they were defined in the specific interface.

14.6 IDL exceptions

An IDL exception shall be mapped to an MHEG-SIR `ExceptionDescription` within a package declaration.

14.6.1 Exception name

The IDL global name of the exception shall be mapped to the MHEG-SIR `ExceptionName` field of this exception description.

14.6.2 Exception members

Members of an IDL exception shall be mapped to the `ExceptionParametersDescription` of this exception description. In this parameters description, each IDL member type shall be mapped to the ParameterType field which identifies a type declared according to the type mapping rules defined in this clause.

14.7 IDL types

An IDL type shall be mapped to an MHEG-SIR `TypeDeclaration`. A type declaration shall have a global scope in the interchanged script.

IDL basic types and constructors shall be mapped to MHEG-SIR primitive types and constructors as summarised in table 3:

Table 3: Type mapping

IDL	MHEG-SIR
void	void
boolean	boolean
octet	octet
short	short
unsigned short	unsigned short
long	long
unsigned long	unsigned long
float	float
double	double
char	character
string	string
enum	enumerated
any	sequence of octet (see note)
struct	structure
union	union
sequence	sequence
array	array

The type definition shall be mapped to an MHEG-SIR `TypeDescription.` If the IDL type is a basic type or if it has already been the subject of another type declaration, this type description shall consist of a type identifier. Otherwise, it shall be constructed according to the following mapping rules:

- a structure field shall be mapped to its rank in the structure description; its name shall not be preserved;

- a union tag value shall be mapped to its rank in the union description; its name and value shall not be otherwise preserved;

- a multidimensional array shall be mapped to an array of arrays;

- an enumerated field shall be mapped to its rank in the enumerated description; its name shall not be preserved;

- a string type shall be mapped to the MHEG-SIR primitive string type; its maximum length (if present) shall not be preserved.

  NOTE: If the type "any" is used by an IDL specification, the MHEG-SIR does not guarantee the preservation of type information.

14.8 IDL constants

IDL constants shall be mapped to an MHEG-SIR `ConstantDeclaration`. A constant declaration shall have a global scope in the interchanged script.

Annex A (normative): ASN.1 notation (level c)

This annex describes the ASN.1 notation (ISO/IEC 8824-1 [2]) for the syntax of the script-data component of the MHEG script class for MHEG conforming script objects whose script-classification component is "script" and whose script-encoding-identification component within the script-hook component is "MHEG-SIR", according to the values maintained by the MHEG data type registration authority.

Profile of ISOMHEG-sc module

The script-data attribute shall be encoded according to the following restrictions of the ISOMHEG-sc module described in MHEG-1:

- the script-inclusion choice shall always be selected for the script-data component;
- the octetstring choice shall always be selected for the script-inclusion component.

As a consequence, MHEG-S conforming script objects shall have the syntax defined by the SIR-Script-Class subtype thereafter.

```
ISOMHEG-sir-sc {...}
DEFINITIONS AUTOMATIC TAGS ::= BEGIN

IMPORTS
        MHEG-Identifier
                FROM ISOMHEG-ud {joint-iso-itu(2) mheg(19) version(1) useful-mheg-types}
        MHEG-Script-Classification, MHEG-Encoding-Description, MHEG-Script-Catalogue
                FROM ISOMHEG-dtra {joint-iso-itu(2) mheg-datatype-registration-authority(13522)}
        Script-Class
                FROM ISOMHEG-sc {joint-iso-itu(2) mheg(19) version(1) script-class(3)}
;

SIR-Script-Class ::= Script-Class (WITH COMPONENTS
{
        script-classification                   ("script"),
                                                        -- as provided by ISOMHEG-dtra
        script-hook-information (WITH COMPONENTS
                        {script-encoding-identification (WITH COMPONENT
                                {mheg-script-catalogue ("MHEG-SIR") }),
                                                        -- as provided by ISOMHEG-dtra
                          script-encoding-description
                                                        -- as provided by ISOMHEG-dtra
                        }),
        script-data (WITH COMPONENT
                {script-inclusion (WITH COMPONENT
                        {octetstring})
                })
}
END
```

NOTE: There is an intrinsic MHEG-1 compatibility problem linked to the fact that the "octetstring" component in the ISOMHEG-sc module should be of InterchangedScript type (imported from ISOMHEG-sir) instead of OCTET STRING, which does not allow any redefinition. If a generic nature is to be maintained, use of EXTERNAL type instead of OCTETSTRING may be considered.

Definition of ISOMHEG-sir module

Interchanged scripts shall have the syntax described by the ISOMHEG-sir module.

```
-- Module: MHEG-SIR (sir)--
--
-- Revision history:
```

```
-- -----------------
--  0.1, 15 Jul 95, Initial Version, based on ISO 13522-3 CD --
-- ------------
--

ISOMHEG-sir {...}
DEFINITIONS AUTOMATIC TAGS ::= BEGIN

EXPORTS   InterchangedScript;

InterchangedScript        ::=          SEQUENCE
{
        type-declarations                SEQUENCE (SIZE (1.. max-nb-declared-types)) OF
                                         TypeDeclaration OPTIONAL,
         constant-declarations           SEQUENCE (SIZE (1 .. max-nb-constants)) OF
                                         ConstantDeclaration OPTIONAL,
         global-variable-declarations    SEQUENCE (SIZE (1 .. max-nb-global-variables)) OF
                                         GlobalVariableDeclaration OPTIONAL,
         external-package-declarations   SEQUENCE (SIZE (1 .. max-nb-packages)) OF
                                         PackageDeclaration OPTIONAL,
         handler-declarations            SEQUENCE (SIZE (1 .. max-nb-messages)) OF
                                         HandlerDeclaration OPTIONAL,
         routine-declarations            SEQUENCE (SIZE (1 .. max-nb-routines)) OF
                                         RoutineDeclaration OPTIONAL
}

TypeDeclaration           ::=    SEQUENCE
{
        identifier                TypeIdentifier OPTIONAL,
        description               TypeDescription
}

TypeDescription           ::=    CHOICE
{
        type-identifier           TypeIdentifier,
        enumerated-description    EnumeratedDescription,
        sequence-description      SequenceDescription,
        array-description         ArrayDescription,
        structure-description     StructureDescription,
        union-description         UnionDescription
}

EnumeratedDescription     ::=    SEQUENCE (SIZE (0 .. max-size-enumerated)) OF ShortValue;

SequenceDescription       ::=    SEQUENCE
{
        length                    INTEGER (0 .. max-size-sequence),
        item-type-description     TypeDescription
}

ArrayDescription          ::=    SEQUENCE
{
        dimension                 INTEGER (1 .. max-size-array),
        item-type-description     TypeDescription
}

UnionDescription          ::=    SEQUENCE (SIZE (1 .. max-size-union)) OF TypeDescription

StructureDescription      ::=    SEQUENCE (SIZE (1 .. max-size-structure)) OF TypeDescription

ConstantDeclaration       ::=    SEQUENCE
{
```

```
        identifier              DataIdentifier OPTIONAL,
        type                    TypeIdentifier ALL EXCEPT 0,
        value                   ConstantValue
}
ConstantValue           ::=     CHOICE
{
        boolean                 BooleanValue,
        octet                   OctetValue,
        short                   ShortValue,
        long                    LongValue,
        unsigned-short          UnsignedShortValue,
        unsigned-long           UnsignedLongValue,
        float                   FloatValue,
        double                  DoubleValue,
        character               CharacterValue,
        string                  StringValue,
        data-identifier         DataIdentifierValue (WITH COMPONENT
                                {local-variable-identifier ABSENT})
        enumerated              EnumeratedValue,
        sequence                SequenceValue,
        array                   ArrayValue,
        union                   UnionValue,
        structure               StructureValue
}


SequenceValue           ::=     SEQUENCE (SIZE (0 to max-size-sequence)) OF ConstantValue;

ArrayValue          ::=         SEQUENCE (SIZE (1 to max-size-array)) OF ConstantValue;

UnionValue          ::=         SEQUENCE
{
        tag                     INTEGER (0 to < max-size-union),
        value                   ConstantValue
}

StructureValue          ::=     SEQUENCE (SIZE (1 to x-size-structure)) OF ConstantValue;

Global-VariableDeclaration      ::=     SEQUENCE
{
        identifier              DataIdentifier OPTIONAL,
        type                    TypeIdentifier,
        initial-value           ConstantReference OPTIONAL
}

PackageDeclaration      ::=     SEQUENCE
{
        identifier              PackageIdentifier OPTIONAL,
        name                    VisibleString OPTIONAL,
        services                SEQUENCE (SIZE (0 to x-nb-services)) OF ServiceDescription,
        exceptions              SEQUENCE (SIZE (0 to ax-nb-exceptions)) OF ExceptionDescription
}

ServiceDescription ::=              SEQUENCE
{
        identifier              unctionIdentifier OPTIONAL,
        operation-name          VisibleString OPTIONAL,
        calling-mode            ENUMERATED {synchronous (0), asynchronous (1)}
                                DEFAULT (synchronous),
        return-value-type       TypeIdentifier,
        parameters-description  SEQUENCE OF ServiceParameterDescription
}
```

```
ServiceParameterDescription   ::=   SEQUENCE
{
        passing-mode                  ENUMERATED {in (1), out (2), inout (3)} DEFAULT (in),
        type                          TypeIdentifier ALL EXCEPT O
}

ExceptionDescription      ::=         SEQUENCE
{
        identifier                    MessageIdentifier OPTIONAL,
        name                          VisibleString OPTIONAL,
        parameters-description        SEQUENCE OF TypeIdentifier OPTIONAL
}

Handler-Declaration       ::=         SEQUENCE
{
        message-reference             MessageReference
        function-reference            FunctionReference

RoutineDeclaration::=           SEQUENCE
{
        routine-description           RoutineDescription,
        program-code                  OCTET STRING
}

RoutineDescription::=           SEQUENCE
{
        identifier                    FunctionIdentifier OPTIONAL,
        return-value-type             TypeIdentifier,
        parameters-description        SEQUENCE OF RoutineParameterDescription
        local-variable-table          SEQUENCE (SIZE (0 to max-nb-local-variables)) OF
                                      LocalVariableDeclaration
}

RoutineParameterDescription ::=     SEQUENCE
{
        passing-mode                  ENUMERATED {value (1), reference (3)} DEFAULT (value),
        type                          TypeIdentifier ALL EXCEPT O
}

LocalVariableDeclaration ::=        SEQUENCE
{
        identifier                    DataIdentifier OPTIONAL,
        type                          TypeIdentifier,
        initial-value                 ConstantReference OPTIONAL
}

MessageReference          ::=         CHOICE
{
        identifier                    MessageIdentifier
        exception-name                VisibleString,
        exception-reference           ExceptionReference,
        predefined-message            PredefinedMessageIndex
}

ExceptionReference        ::=         SEQUENCE
{
        package-identifier            PackageIdentifier,
        exception-index               ExceptionIndex
}

ConstantReference         ::=         CHOICE
```

```
{
        identifier                      DataIdentifier,
        constant-index                  ConstantIndex,
        value                           ConstantValue
}

FunctionReference::=            CHOICE
{
        identifier                      FunctionIdentifier,
        service-reference               ServiceReference,
        predefined-function             PredefinedFunctionIndex,
        routine-reference               RoutineIndex
}

ServiceReference  ::=                   SEQUENCE
{
        package-identifier              PackageIdentifier,
        service-index                   ServiceIndex
}

max-size-array                  ::=     65 536
max-size-sequence               ::=     65 535
max-size-union                  ::=     256
max-size-structure              ::=     256
max-size-enumerated             ::=     256
max-nb-global-variables         ::=     28 672
max-nb-constants                ::=     4 096
max-nb-local-variables          ::=     32 768
max-nb-data             ::=     65 536
        -- max-nb-constants+max-nb-global-variables+max-nb-local-variables
max-nb-packages                 ::=     224
max-nb-services                 ::=     256
max-nb-routines                 ::=     4 096
max-nb-predef-functions         ::=     4 096
max-nb-functions                ::=     65 536
        -- max-nb-packagesxmax-nb-services+max-nb-predef-functions+max-nb-routines
max-nb-exceptions       ::=     256
max-nb-predef-messages          ::=     8 192
max-nb-messages                 ::=     65 536
        -- max-nb-packagesxmax-nb-exceptions+max-nb-predef-messages
max-nb-declared-types           ::=     28 672
max-nb-predef-types             ::=     4 096
max-nb-types                    ::=     32 768
        -- max-nb-predef-types + max-nb-declared-types

BooleanValue                    ::=     BOOLEAN
OctetValue                      ::=     OCTET STRING (SIZE 1)
EnumeratedValue                 ::=     INTEGER (0 to < max-size-enumerated)
ShortValue                      ::=     INTEGER (-32 768 to 32 767)
LongValue                       ::=     INTEGER (-2 147 483 648 to 2 147 483 647)
UnsignedShortValue              ::=     INTEGER (0 to 65535)
UnsignedLongValue               ::=     INTEGER (0 to 4 294 967 295)
FloatValue                      ::=     REAL
DoubleValue                     ::=     REAL
CharacterValue                  ::=     BMPString (SIZE 1)
StringValue                     ::=     BMPString

TypeIdentifier                  ::=     INTEGER (0 to < max-nb-types)
DataIdentifier                  ::=     INTEGER (0 to < max-nb-data)
FunctionIdentifier              ::=     INTEGER (0 to < max-nb-functions)
MessageIdentifier               ::=     INTEGER (0 to < max-nb-messages)
PackageIdentifier               ::=     INTEGER (0 to < max-nb-packages)
```

```
ConstantIndex                ::=  INTEGER (0 to < max-nb-constants)
GlobalVariableIndex          ::=  INTEGER (0 to < max-nb-global-variables)
LocalVariableIndex      ::=  INTEGER (0 to < max-nb-local-variables)
ServiceIndex                 ::=  INTEGER (0 to < max-size-services)
RoutineIndex                 ::=  INTEGER (0 to < max-nb-routines)
PredefinedFunctionIndex      ::=  INTEGER (0 to < max-nb-predef-functions)
ExceptionIndex               ::=  INTEGER (0 to < max-nb-exceptions)
PredefinedMessageIndex::=  INTEGER (0 to < max-nb-predef-messages)

END
```

Annex B (normative): Coded representation (level d)

Coding for interchanged scripts

Interchanged scripts shall be encoded according to ASN.1 Distinguished Encoding Rules (DER) (ISO/IEC 8825-1 [3].

NOTE: This is meant to make the MHEG-S engine's decoding task as efficient as possible by removing all ASN.1 encoding options that might delay or complicate it.

Coding for the program code

The value of the program-code component of the RoutineDeclaration type defined by ISOMHEG-sir (see annex A) shall be encoded according to the following rules.

The sequence of instructions that make up the program code of a routine shall be encoded as a sequence of octets. The order of encoding will be the same as the order in which the instructions are intended to be executed.

Each instruction shall be encoded using one octet for the op-code, followed by zero to three octets for the operands, depending on the op-code.

The op-codes shall be encoded using the bitstring defined by table B.1:

- the two most significant bits (i.e. bits 8 and 7) represent the number of following octets to be taken as operands for the instruction:

  a) 00: 0 octet;
  b) 01: 1 octet;
  c) 10: 2 octets;
  d) 11: 3 octets.

- the two next most significant bits (i.e. bits 6 and 5) represent the category:

  a) 0000: control flow;
  b) 0100: control flow (jumps);
  c) 1000: control flow (long jumps, calls);
  d) 0001: Arithmetic operators;
  e) 0010: logical operators;
  f) 0110: logical operators (shift);
  g) 0011: comparison operators;
  h) 0101: stack manipulation (drop);
  i) 1001: stack manipulation (pushi);
  j) 1010: variable use (push);
  k) 1110: variable use (get);
  l) 1011: variable assignment (pop, inc/dec);
  m) 1111: variable assignment (set).

According to the op-code of the instruction, the operands shall have the length and encoding specified by table XX.

"Data identifier" operands shall be encoded using two octets as follows:

- if bit 16 is "1", the data identifier is meant to reference a local variable, where bits 15 to 1 represent the local variable index (from 0 to 32 767);

- if bits 16 to 13 are "0000", the data identifier is meant to reference a constant, where bits 12 to 1 represent the constant index (from 0 to 4 095);

- otherwise, the data identifier is meant to reference a global variable, where bits 15 to 1 represent the global variable index (from 0 to 28 671) incremented by 4 096.

"Function identifier" operands shall be encoded on two octets as follows:

- if bits 16 to 13 are "000", the function identifier is meant to reference a routine, where bits 12 to 1 represent the routine index (from 0 to 4 095);

- if bits 16 to 13 are "111", the function identifier is meant to reference a predefined function (MHEG-API operation), where bits 12 to 1 represent the predefined function index (from 0 to 4 095);

- otherwise, the function identifier is meant to reference a service, where bits 16 to 9 represent the package identifier (from 0 to 223) incremented by 16, and where bits 8 to 1 represent the service index (from 0 to 255) within this package.

1-octet "offset" operands shall be encoded in complement-to-one notation on 1 octet: bit 8 represents the direction of movement, bits 7 to 1 represent the number of units to shift in that direction.

2-octet "offset" operands shall be encoded in complement-to-one notation on 2 octets: bit 16 represents the direction of movement, bits 15 to 1 represent the number of units to shift in that direction.

"Value" operands shall be encoded in complement-to-two notation on two octets, for interpretation as signed integer values.

"Index" operands shall be encoded on one octet, for interpretation as unsigned integer values.

Table B.1: Encoding of MHEG-SIR instructions

Instruction mnemonics	Op-code	Op1 length	Op1 encoding	Op2 length	Op2 encoding
NOP	0000 0000	0			
YIELD	0000 0010	0			
RET	0000 0011	0			
ADD	0001 0000	0			
SUB	0001 0001	0			
MUL	0001 0010	0			
DIV	0001 0011	0			
REM	0001 0100	0			
NEG	0001 0101	0			
NOT	0010 0000	0			
AND	0010 0001	0			
OR	0010 0010	0			
XOR	0010 0011	0			
EQ	0011 0000	0			
NEQ	0011 0001	0			
LE	0011 0100	0			
LT	0011 0101	0			
GE	0011 0110	0			
GT	0011 0111	0			
JT	0100 0000	1	(signed) offset		
JF	0100 0001	1	(signed) offset		
JMP	0100 0010	1	(signed) offset		
DROP	0101 0000	1	(unsigned) index		
SHIFT	0110 0000	1	(signed) offset		
LJT	1000 0000	2	(signed) offset		
LJF	1000 0001	2	(signed) offset		
LJMP	1000 0010	2	(signed) offset		
CALL	1000 1000	2	function identifier		
XCALL	1000 1001	2	function identifier		
PUSHI	1001 0000	2	(signed) value		
PUSH	1010 0000	2	data identifier		
PUSHR	1010 0001	2	data identifier		
POP	1011 0000	2	data identifier		
POPR	1011 0001	2	data identifier		
POPC	1011 0010	2	data identifier		
INC	1011 0100	2	data identifier		
DEC	1011 0101	2	data identifier		
GET	1110 0000	3	data identifier	1	(unsigned) index
SET	1111 0000	3	data identifier	1	(unsigned) index
SETC	1111 0010	3	data identifier	1	(unsigned) index

Annex C (normative): MHEG-SIR predefined elements

This annex lists the predefined types, functions and messages of the MHEG-SIR, together with their corresponding indices.

Predefined types, functions and messages may be referenced by their identifier and used within interchanged scripts, the same way types, functions and messages declared within the global declarations part of interchanged scripts would.

Predefined types

MHEG-SIR predefined types consist of:

- primitive types;
- MHEG-API types.

Primitive types

The primitive types defined by this ETS shall be encoded using predefined type identifiers as listed in table C.1.

Table C.1: Predefined type identifiers for primitive types

Type name	Type identifier
void	0
boolean	1
octet	2
short	3
long	4
unsigned short	5
unsigned long	6
float	7
double	8
char	9
string	10
data identifier	11
object reference	12

All types that can be expressed in MHEG-SIR (including predefined MHEG types) can be built using the MHEG-SIR primitive types and the following constructors:

- array
- sequence
- structure
- union
- enumerated

MHEG-API types

The MHEG-API types defined by the MHEG-API (ITU-T Recommendation T.177 [4]) shall be encoded using predefined type identifiers which shall be listed using the structure in table C.2.

NOTE: MHEG-API types are intended for use by interchanged scripts to express information which is exchanged between the script interpreter and the MHEG engine.

Table C.2: Predefined type identifiers for MHEG-API types

Type name	Type identifier

The IDL definition of these types, as provided by the MHEG-API (ITU-T Recommendation T.177 [4]), shall be mapped to MHEG-SIR type descriptions using the IDL mapping rules described in clause 13.

Predefined functions

MHEG-SIR predefined functions consist of:

- MHEG-API operations.

The MHEG-API operations defined by the MHEG-API (ITU-T Recommendation T.177 [4]) shall be encoded using predefined function identifiers which shall be listed using the structure in table C.3.

Table C.3: Predefined function identifiers for MHEG-API operations

Operation name	Predefined function index	Function identifier

The IDL definition of these operations, as provided by the MHEG-API (ITU-T Recommendation T.177 [4]), shall be mapped to MHEG-SIR function descriptions using the IDL mapping rules described in clause 13.

Predefined messages

MHEG-SIR predefined messages consist of:

- MHEG actions targeted at an rt-script;
- MHEG-API exceptions.

MHEG actions targeted at rt-scripts

The MHEG actions targeted at rt-scripts shall be encoded using predefined message identifiers which shall be listed using the structure in table C.4.

Table C.4: Predefined message identifiers for MHEG-API actions

Action name	Predefined message index	Message identifier

MHEG-API exceptions

The MHEG-API exceptions defined by the MHEG-API (ITU-T Recommendation T.177 [4]) shall be encoded using predefined message identifiers which shall be listed using the structure in table C.5.

Table C.5: Predefined message identifiers for MHEG-API exceptions

Operation name	Predefined message index	Message identifier

The IDL definition of these exceptions, as provided by the MHEG-API (ITU-T Recommendation T.177 [4]), shall be mapped to MHEG-SIR message descriptions using the IDL mapping rules described in clause 13.

Annex D (normative): IDL Platform mapping specification form

MHEG-S engines shall allow access to the services provided by the run-time environment of a given platform, provided this run-time environment complies with the registered platform mapping specification for this platform.

The registered platform mapping specifications shall be provided according to the template described in this annex, with all fields being completed.

Platform-mapping specification form

This MHEG-SIR platform-mapping specification defines the mechanisms that shall be used by MHEG-S engines to access the services provided by the run-time environment on the platform.

Platform description

The platform to which this specification applies is <platform_description>.

Package availability procedure

To know whether an IDL specification is available within the run-time environment and to locate it, an MHEG-S engine shall proceed as follows: <package_availability_procedure>

Package load procedure

To make the operations of an available IDL specification accessible, an MHEG-S engine shall proceed as follows: <package_load_procedure>

Package unload procedure

To stop the operations of an available IDL specification from being accessible, an MHEG-S engine shall proceed as follows: <package_unload_procedure>

Operation invocation procedure

To invoke an operation of an accessible IDL specification, an MHEG-S engine shall proceed as follows: <operation_invocation_procedure>

Parameter passing procedure

When invoking an IDL operation, an MHEG-S engine shall pass "in" parameters as follows: <in_parameter_passing _procedure>

When invoking an IDL operation, an MHEG-S engine shall pass "out" parameters as follows: <out_parameter_passing _procedure>

When invoking an IDL operation, an MHEG-S engine shall pass "inout" parameters as follows: <inout_parameter_passing _procedure>

Output parameter retrieval procedure

To retrieve the values of "out" or "inout" parameters after invoking an IDL operation, an MHEG-S engine shall proceed as follows: <output_parameter_retrieval_procedure>

Return value retrieval procedure

To retrieve the return value of a previously invoked IDL operation, an MHEG-S engine shall proceed as follows: <return_value _retrieval_procedure>

Data encoding rules

The values of data that are interchanged between the MHEG-S engine and the run-time environment shall be encoded as follows: <data_encoding_rules>

Exception retrieval procedure

To retrieve exceptions that are raised by the run-time environment, an MHEG-S engine shall proceed as follows: <exception_retrieval_procedure>

System exceptions

The system exceptions that may be raised by the run-time environment and retrieved by an MHEG-S engine shall be defined as follows: <system_exception_definitions>

Resource limitations

When using the run-time environment on the platform, the following resource limitations apply: <resource_limitations_statement>

Annex E (informative): EBNF notation for MHEG-SIR syntax

This is the description of the syntax of MHEG-SIR interchanged scripts; it is used as a level b) variant which maps to the ASN.1 description provided in annex A.

Conventions:

- non-terminals are written as normal text;
- terminal types are written in uppercase;
- literal terminals are enclosed in single quotes.

// Structure

```
InterchangedScript       ::=    TypeDeclaration*
                                ConstantDeclaration*
                                GlobalVariableDeclaration*
                                PackageDeclaration*
                                HandlerDeclaration*
                                RoutineDeclaration*
```

// Type declarations

```
TypeDeclaration          ::=    TypeIdentifier?
                                TypeDescription

TypeDescription          ::=    TypeIdentifier
                         /      EnumeratedDescription
                         /      SequenceDescription
                         /      ArrayDescription
                         /      StructureDescription
                         /      UnionDescription

EnumeratedDescription ::=    INTEGER*          // List of values

SequenceDescription      ::=    INTEGER?          // Sequence (max) size
                                TypeDescription

ArrayDescription         ::=    INTEGER       // Array dimension
                                TypeDescription

UnionDescription         ::=    TypeDescription+

StructureDescription     ::=    TypeDescription+
```

// Data declarations

```
ConstantDeclaration      ::=    DataIdentifier?
                                TypeIdentifier
                                ConstantValue

ConstantValue            ::=    BOOLEAN
                         /      OCTET
                         /      INTEGER       // all numeric types
                         /      REAL              // float or double
                         /      STRING            // character or string
                         /      DataIdentifier
                         /      ConstantValue*// sequence, array or structure
                         /      UnionValue

UnionValue               ::=    INTEGER       // Tag index
                                ConstantValue

GlobalVariableDeclaration ::=   DataIdentifier?
                                TypeIdentifier
                                ConstantReference? // Initial value

ConstantReference        ::=    DataIdentifier
                         /      INTEGER       // constant index
                         /      ConstantValue
```

```
// Package declarations

PackageDeclaration   ::=       PackageIdentifier?
                               VisibleString     // Package name
                               ServiceDescription*
                               ExceptionDescription*

ServiceDescription   ::=       FunctionIdentifier?
                               VisibleString?// IDL global name
                               CallingMode?
                               TypeIdentifier// return value
                               ParameterDescription*

ServiceParameterDescription  ::= ServicePassingMode?
                               TypeIdentifier

CallingMode          ::=    "SYNCHRONOUS" / "ASYNCHRONOUS"
ServicePassingMode   ::=    "IN" / "OUT" / "INOUT"

ExceptionDescription ::=    MessageIdentifier?
                            VisibleString?//IDL exception global name
                            TypeIdentifier*   //Parameter types

// Handler declarations

HandlerDeclaration   ::=    MessageReference
                            FunctionReference

MessageReference     ::=    MessageIdentifier
                     /      VisibleString // exception name
                     /      ExceptionReference
                     /      INTEGER    // predefined message index

ExceptionReference   ::=    PackageIdentifier
                            INTEGER    // exception index

FunctionReference    ::=    FunctionIdentifier
                     /      ServiceReference
                     /      INTEGER    // routine index,
                                       // predefined message index

ServiceReference     ::=    PackageIdentifier
                            INTEGER    // service index

// Routine declarations

RoutineDeclaration   ::=    FunctionIdentifier?
                            TypeIdentifier    // for return value
                            RoutineParameterDescription*
                            LocalVariableDeclaration*
                            OCTET STRING      // program code

RoutineParameterDescription ::=    RoutinePassingMode?
                                   TypeIdentifier

RoutinePassingMode   ::=    "VALUE" / "REFERENCE"

LocalVariableDeclaration ::=    DataIdentifier?
                                TypeIdentifier
                                ConstantReference? // initial value (constant)

// Useful definitions

TypeIdentifier       ::=    INTEGER
DataIdentifier       ::=    INTEGER
FunctionIdentifier   ::=    INTEGER
MessageIdentifier    ::=    INTEGER
PackageIdentifier    ::=    INTEGER
```

Annex F (informative): Textual notation for MHEG-SIR programs

This notation can also be used as a textual form for the human-readable notation of MHEG-SIR scripts. It may be used for expressing MHEG-SIR examples in human-readable form. It is given using EBNF notation.

Conventions:

- non-terminals are written as normal text;
- terminal types are written in uppercase;
- literal terminals are enclosed in single quotes;
- integers should be expressed in decimal (as -XXX) or hexadecimal form (as XXXXh);
- reals should be expressed as -XXXe-XX;
- all elements should be separated by at least one blank or line feed character.

```
InterchangedScript   ::=       "SCRIPT"
                               TypeDeclaration*
                               ConstantDeclaration*
                               GlobalVariableDeclaration*
                               PackageDeclaration*
                               HandlerDeclaration*
                               RoutineDeclaration*
                               "ENDSCRIPT"
```

// Type declarations

```
TypeDeclaration      ::=       "TYPE"
                               {"ID" TypeIdentifier}?
                               TypeDescription
                               "ENDTYPE"

TypeDescription      ::=    TypeIdentifier
                     /      "ENUM" EnumeratedDescription "ENDENUM"
                     /      "SEQUENCE"    SequenceDescription "ENDSEQUENCE"
                     /      "ARRAY"    ArrayDescription "ENDARRAY"
                     /      "STRUCT"   StructureDescription "ENDSTRUCT"
                     /      "UNION"UnionDescription "ENDUNION"

EnumeratedDescription    ::=    INTEGER*          // List of values

SequenceDescription      ::=    INTEGER       // Sequence (max) size
                                TypeDescription

ArrayDescription         ::=    INTEGER       // Array dimension
                                TypeDescription

UnionDescription     ::=    TypeDescription+

StructureDescription ::=    TypeDescription+
```

// Data declarations

```
ConstantDeclaration  ::=    "CONSTANT"
                            {"ID" DataIdentifier}?
                            TypeIdentifier
                            ConstantValue
                            "ENDCONSTANT"

ConstantValue        ::=    "BOOLEAN" BOOLEAN
                     /      "OCTET" OCTET
                     /      "SHORT" INTEGER
                     /      "LONG" INTEGER
                     /      "UNSIGNED SHORT" INTEGER
                     /      "UNSIGNED LONG" INTEGER
                     /      "FLOAT" REAL
                     /      "DOUBLE" REAL                          /   "CHAR" STRING
                     /      "STRING" STRING
                     /      "ID" DataIdentifier
                     /      "SEQUENCE" ConstantValue*
```

```
                    /      "STRUCT" ConstantValue*
                    /      "ARRAY" ConstantValue*
                    /      "UNION" UnionValue

UnionValue        ::=      INTEGER          // Tag index
                           ConstantValue

GlobalVariableDeclaration   ::=   "VARIABLE"
                                  {"ID" DataIdentifier}?
                                  TypeIdentifier
                                  ConstantReference? // Initial value
                                  "ENDVARIABLE"

ConstantReference   ::=    DataIdentifier
                    /      "INDEX" INTEGER      // constant index
                    /      ConstantValue
```

// Package declarations

```
PackageDeclaration  ::=    "PACKAGE"
                           {"ID" PackageIdentifier}?
                           VisibleString // Package name
                           ServiceDescription*
                           ExceptionDescription*
                           "ENDPACKAGE"

ServiceDescription  ::=    "SERVICE"
                           {"ID" FunctionIdentifier}?
                           VisibleString?// IDL global name
                           CallingMode?
                           TypeIdentifier// return value
                           ParameterDescription*
                           "ENDSERVICE"

ServiceParameterDescription ::=    "PARAM"
                                   ServicePassingMode
                                   TypeIdentifier

CallingMode         ::=    "SYNC" / "ASYNC"
ServicePassingMode  ::=    "IN" / "OUT" / "INOUT"

ExceptionDescription ::=   "EXCEPTION"
                           {"ID" MessageIdentifier}?
                           VisibleString?//IDL exception global name
                           TypeIdentifier*   //Parameter types
                           "ENDEXCEPTION"
```

// Handler declarations

```
HandlerDeclaration  ::=    "HANDLER"
                           MessageReference
                           FunctionReference

MessageReference    ::=    MessageIdentifier
                    /      "NAME" VisibleString // exception name
                    /      "REF" ExceptionReference
                    /      "PREDEF" INTEGER  // predefined message index

ExceptionReference  ::=       PackageIdentifier
                              INTEGER    // exception index

FunctionReference   ::=       FunctionIdentifier
                    /   "     REF" ServiceReference
                    /   "     ROUTINE" INTEGER     // routine index,
                    /         "PREDEF" INTEGER  // predefined message index

ServiceReference    ::=       PackageIdentifier
                              INTEGER    // service index
```

// Routine declarations

```
RoutineDeclaration  ::=       "ROUTINE"
```

```
                              {"ID" FunctionIdentifier}?
                              TypeIdentifier    // for return value
                              RoutineParameterDescription*
                          LocalVariableDeclaration*
                          Instruction+

RoutineParameterDescription ::=    "PARAM"
                                   RoutinePassingMode?
                                   TypeIdentifier

RoutinePassingMode   ::=           "VAL" / "VAR"

LocalVariableDeclaration ::=       "VARIABLE"
                                   {"ID" DataIdentifier}?
                                   TypeIdentifier
                                   ConstantReference? // initial value (constant)
```

// Useful definitions

```
TypeIdentifier       ::=    INTEGER
DataIdentifier       ::=    INTEGER
FunctionIdentifier   ::=    INTEGER
MessageIdentifier    ::=    INTEGER
PackageIdentifier    ::=    INTEGER
```

// Program code

```
Instruction          ::=    StackOperator
                     /      ShortUnaryInstruction ShortOffset
                     /      LongUnaryInstruction LongOffset
                     /      "DROP" Index
                     /      AssignmentInstruction DataIdentifier
                     /      BinaryInstruction DataIdentifier Index
                     /      CallInstruction FunctionIdentifier

StackOperator     ::=       "ADD"  // arithmetic
                  /         "SUB"
                  /      "MUL"
                  /      "DIV"
                  /      "REM"
                  /      "NEG"
                  /      "AND"      // logical
                  /      "OR"
                  /      "XOR"
                  /      "NOT"
                  /      "EQ"       // comparison
                  /      "NEQ"
                  /      "GT"
                  /      "GE"
                  /      "LT"
                  /      "LE"
                  /      "YIELD"// control flow
                  /      "RET"
                  /      "NOP"

AssignmentInstruction    ::=    "INC"
                         /      "DEC"
                         /      "PUSHR"
                         /      "PUSH"
                         /      "POPR"
                         /      "POP"
                         /      "POPC"

ShortUnaryInstruction::=    "SHIFT"
                     /      "JT"
                     /      "JF"

LongUnaryInstruction  ::=    "LJT"
                     /      "LJF"
                     /      "PUSHI"

BinaryInstruction    ::=    "SET"
                     /      "SETC"
```

```
                        /       "GET"

CallInstruction         ::=     "CALL"
                        /       "XCALL"

ShortOffset         ::=         INTEGER
LongOffset          ::=         INTEGER
Index               ::=         INTEGER
```

Annex G (informative): MHEG entities

(reproduced from ETS 300 714 [10]).

MHEG objects

According to ISO/IEC 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 13522-1 [1], subclause 5.2.4. 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 13522-1 [1]. The MHEG identifier of an MHEG object shall 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. ISO/IEC 13522-1 [1] imposes a constraint on the design of applications using MHEG 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.

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 13522-1 [1], subclause 5.2.4. 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 allows the guarantee that all mh-objects shall be identifiable.

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

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 13522-1 [1], subclause 5.2.4. 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 13522-1 [1]. 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 used as models for rt-objects are non-identifiable, i.e. have no MHEG identifier. This allows the guarantee that all rt-objects shall be identifiable.

Rt-objects may be referenced using MHEG generic references.

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.

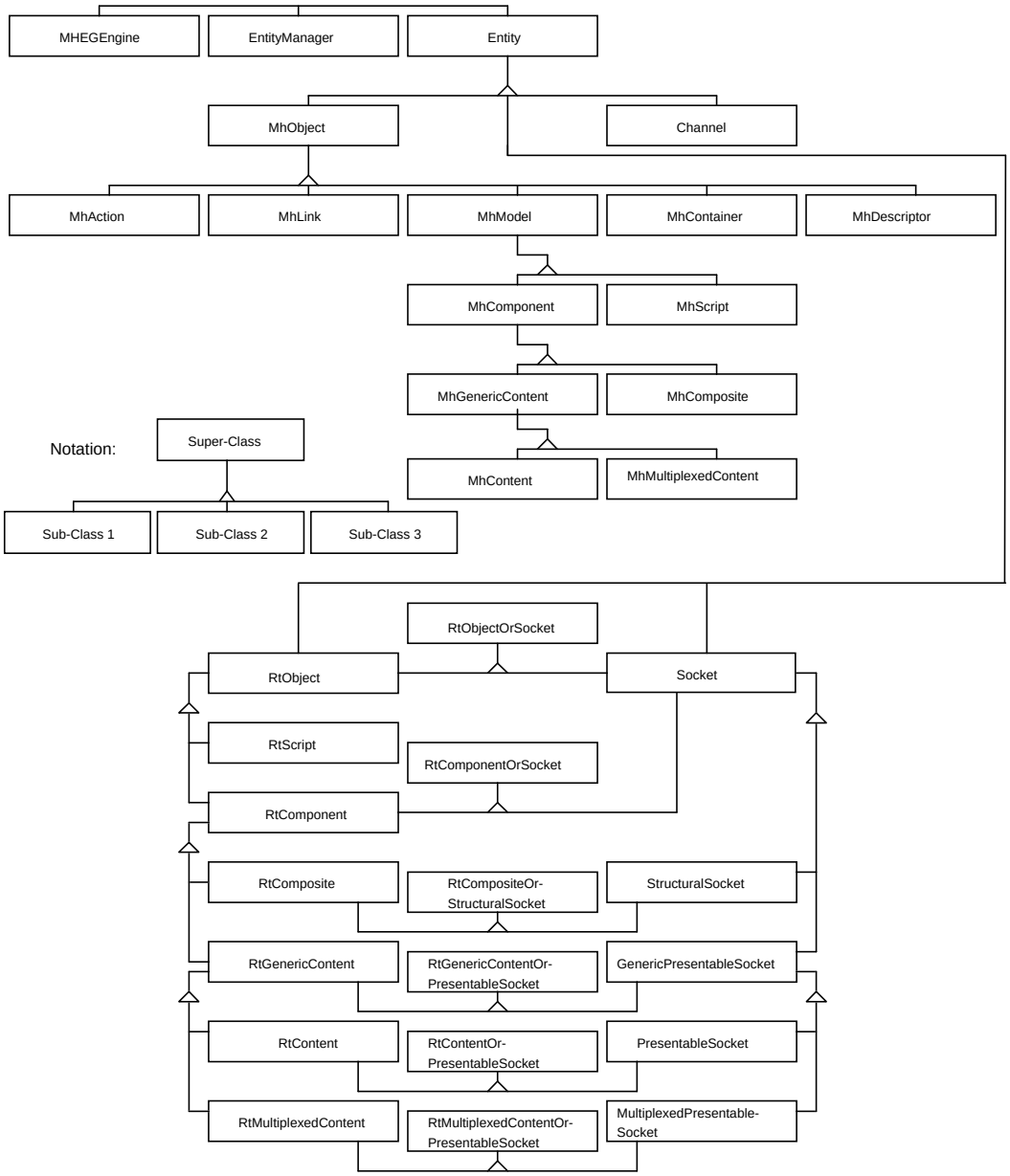
Stored MHEG objects are representations of MHEG objects which are usually located in files or database records. For instance, one given MHEG object (i.e. bitstring) may be stored in many places, i.e. represented by many stored MHEG objects. Such locations are usually identified using file names or database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

Annex H (informative): MHEG Application Programming Interface (MHEG-API)

Object model

This clause 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 13522-1 [1], subclause 5.2.4, i.e. objects available to the MHEG engine.

NOTE 2: RtObjects (and their subtypes) match form c) objects as defined in ISO/IEC 13522-1 [1], subclause 5.2.4, i.e. instances of MhObjects available to the presentation process.

Figure H.1: Object model

H.1 IDL specification of the MHEG-API

H.1.1 `MHEGEngine` object

This subclause defines the operations of the `MHEGEngine` object.

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 invokation shall reinitialise the MHEG Engine.

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.

IDL description:

```
interface MHEGEngine {

    void   initialiseEngine();
    void   shutdownEngine();

};
```

H.1.2 `NotificationManager` object

This subclause defines the operations of the `NotificationManager` object.

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.

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 or not the action was completed (with a default value assigned to the inadequate parameter). The `parameter_number` member identifies the rank of the invalid parameter.

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);

};
```

H.1.3 `EntityManager` object

This subclause defines the operations of the `EntityManager` object.

getAvailableMhObjects operation

Synopsis:

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

Description:

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

A 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.

getAvailableRtObjects operation

Synopsis:

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

Description:

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

A 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.

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.

releaseAlias operation

Synopsis:

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

Description:

This operation enables the release of 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 or not the action was completed (with a default value assigned to the inadequate parameter). The `parameter_number` member identifies the rank of the invalid parameter.

IDL description:

```
interface  EntityManager {

    sequence<MHEGIdentifier>
        getAvailableMhObjects();

    sequence<RtObjectIdentifier>
        getAvailableRtObjects();

    sequence<ChannelIdentifier>
        getAvailableChannels();

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


};
```

H.1.4 `Entity` object

This subclause defines the operations of the `Entity` object.

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 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.

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.

IDL description:

```
interface Entity  {

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

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

H.1.5 `MhObject` object

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

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.

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.

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 a 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 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.

Destroy operation

Synopsis:

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

Description:

This operation enables the removing of a 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 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.

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 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.

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.

Kill operation

Synopsis:

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

Description:

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

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();

};
```

H.1.6 `MhAction` object

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

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 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 or not the action was completed (with a default value assigned to the inadequate parameter). The `parameter_number` member identifies the rank of the invalid parameter.

IDL description:

```
interface  MhAction: MhObject   {

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

};
```

H.1.7 `MhLink` object

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

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 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.

IDL description:

```
interface  MhLink: MhObject  {

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

H.1.8 `MhModel` object

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

IDL description:

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

H.1.9 `MhComponent` object

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

IDL description:

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

H.1.10 `MhGenericContent` object

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

copy operation

Synopsis:

Interface:	`MhContent`	
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 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 or not the action was completed (with a default value assigned to the inadequate parameter). The `parameter_number` member identifies the rank of the invalid parameter.

IDL description:

```
interface  MhGenericContent: MhComponent   {

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

H.1.11 `MhContent` object

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

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 the storing or modification of 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 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 or not the action was completed (with a default value assigned to the inadequate parameter). The `parameter_number` member identifies the rank of the invalid parameter.

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 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 or not the action was completed (with a default value assigned to the inadequate parameter). The `parameter_number` member identifies the rank of the invalid parameter.

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);

};
```

H.1.12 `MhMultiplexedContent` object

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

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 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 or not the action was completed (with a default value assigned to the inadequate parameter). The `parameter_number` member identifies the rank of the invalid parameter.

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. an 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 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 or not the action was completed (with a default value assigned to the inadequate parameter). The `parameter_number` member identifies the rank of the invalid parameter.

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);

};
```

H.1.13 `MhComposite` object

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

IDL description:

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

H.1.14 `MhScript` object

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

IDL description:

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

H.1.15 `MhContainer` object

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

IDL description:

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

H.1.16 `MhDescriptor` object

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

IDL description:

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

H.1.17 `RtObjectOrSocket` object

This subclause defines the operations of the `RtObjectOrSocket` object.

setGlobalBehaviour operation

Synopsis:

Interface:	`RtObject`	
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 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 or not the action was completed (with a default value assigned to the inadequate parameter). The `parameter_number` member identifies the rank of the invalid parameter.

getGlobalBehaviour operation

Synopsis:

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

Description:

This operation retrieves all the value attributes 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 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.

run operation

Synopsis:

Interface: `RtObject`
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 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.

stop operation

Synopsis:

Interface: `RtObject`
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 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.

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);

};
```

H.1.18 `RtObject` object

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

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.

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.

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 a 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 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.

delete operation

Synopsis:

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

Description:

This operation enables the removal of a 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 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.

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 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.

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.

kill operation

Synopsis:

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

Description:

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

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 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.

IDL description:

```
interface  RtObject: Entity  {

    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);

};
```

H.1.19 `Socket` object

This subclause defines the operations of the `Socket` object. The object inherits from the `RtObjectOrSocket` and from the `Entity` object.

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.

unbind operation

Synopsis:

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

Description:

This operation cancels the bindind 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.

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.

kill operation

Synopsis:

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

Description:

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

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 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.

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 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 or not the action was completed (with a default value assigned to the inadequate parameter). The `parameter_number` member identifies the rank of the invalid parameter.

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 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, the computation of its result and the error conditions that cause exceptions to be raised are defined by ISO/IEC 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.

IDL description:

```
interface  Socket: Entity, RtObjectOrSocket    {

    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);

};
```

H.1.20 `RtScript` object

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

setParameters operation

Synopsis:

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

Description:

This operation enables to pass parameters 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 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.

getTerminationStatus operation

Synopsis:

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

Description:

This operation get 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 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.

IDL description:

```
interface  RtScript: RtObject   {

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

    TerminationStatusValue
        getTerminationStatus()
    raises(InvalidTarget);

};
```

H.1.21 `RtComponentOrSocket` object

This subclause defines the operations of the `RtComponentOrSocket` object.

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 an 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 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.

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 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.

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 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.

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 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.

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 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.

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 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.

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 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.

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 retrieves the presentation priority 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 13522-1 [1], subclause 51.3.3.

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 or not the action was completed (with a default value assigned to the inadequate parameter). The `parameter_number` member identifies the rank of the invalid parameter.

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 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.

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.

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 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 or not the action was completed (with a default value assigned to the inadequate parameter). The `parameter_number` member identifies the rank of the invalid parameter.

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 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.

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 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 or not the action was completed (with a default value assigned to the inadequate parameter). The `parameter_number` member identifies the rank of the invalid parameter.

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 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 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.

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 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.

getVDLength operation

Synopsis:

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

Description:

This operation retrieves the Visible Duration (VD) length value of the rt-component or socket either in OGTU or in 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 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.

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 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.

getCurrentTemporalPosition operation

Synopsis:

Interface:	`x`
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 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.

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 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 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.

getOGTR operation

Synopsis:

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

Description:

This operation retrieves the 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 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.

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 caculate 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 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.

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 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.

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 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.

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 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 or not the action was completed (with a default value assigned to the inadequate parameter). The `parameter_number` member identifies the rank of the invalid parameter.

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 CGS (through the chain of mappings OGS-RGS), the ratio between the PS in OGSU, i.e. the OGS lengths, and the projection in 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 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.

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 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.

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 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.

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 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.

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 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.

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: PSAP, VSIAP or 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 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.

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 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 or not the action was completed (with a default value assigned to the inadequate parameter). The `parameter_number` member identifies the rank of the invalid parameter.

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 szes 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 13522-1 [1], subclause 53.2.10.

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

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

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

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 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.

setResizingAbility operation

Synopsis:

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

Description:

This operation specifies whether the user is able or not to resize 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 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.

setScalingAbility operation

Synopsis:

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

Description:

This operation specifies whether the user is able or not to scale 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 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.

setScrollingAbility operation

Synopsis:

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

Description:

This operation specifies whether the user is able to scroll or not 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 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.

getGSR operation

Synopsis:

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

Description:

This operation retrieves the GSR value of the OGS of the rt-component or socket. This ratio defined the number of OGSU which are to be mapped in one 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 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.

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 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.

getAspectRatio operation

Synopsis:

Interface:	x
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 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.

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 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.

getVSGS operation

Synopsis:

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

Description:

This operation retrieves the 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 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.

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 VSGSU.

The `getVS` operation triggers the execution of the "get VS" 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 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.

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 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.

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 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.

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 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.

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 positionned the VSIAP relatively to the PSAP.

The `getVSIAPPosition` operation triggers the 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 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.

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 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.

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 relatively to a refernce point. This is used to positionned the VSEAP relatively 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 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.

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 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.

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 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.

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 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.

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 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.

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 parent of the rt-component or socket.

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 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 or not the action was completed (with a default value assigned to the inadequate parameter). The `parameter_number` member identifies the rank of the invalid parameter.

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 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.

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 of 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 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.

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 13522-1 [1], subclause 57.3.1.

The `min_mumber_of_selections` parameter specifies the value of the "min mumber of selections" parameter of the "get selectability" action.

The `max_mumber_of_selections` parameter specifies the value of the "max mumber 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

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 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.

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 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.

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 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.

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 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.

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 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

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 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.

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 of 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 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.

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 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.

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 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.

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 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.

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 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.

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 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.

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 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.

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);

};
```

H.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.

IDL description:

```
interface  RtComponent: RtComponentOrSocket, RtObject {};
```

H.1.23 `RtCompositeOrStructuralSocket` object

This subclause defines the operations of the `RtCompositeOrStructuralSocket` object.

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 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 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.

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 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.

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 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 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.

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 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.

getNumberOfSelectedSockets operation

Synopsis:

```
.               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 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.

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 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.

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 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 or not the action was completed (with a default value assigned to the inadequate parameter). The `parameter_number` member identifies the rank of the invalid parameter.

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 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 or not the action was completed (with a default value assigned to the inadequate parameter). The `parameter_number` member identifies the rank of the invalid parameter.

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);

};
```

H.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.

IDL description:

```
interface  RtComposite: RtCompositeOrStructuralSocket, RtComponent {};
```

H.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.

IDL description:

```
interface  StructuralSocket: RtCompositeOrStructuralSocket, Socket {};
```

H.1.26 `RtGenericContentOrPresentableSocket` object

This subclause defines the operations of the `RtGenericContentOrPresentableSocket` object.

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 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 or not the action was completed (with a default value assigned to the inadequate parameter). The `parameter_number` member identifies the rank of the invalid parameter.

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 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.

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 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.

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 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.

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 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.

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, 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 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 or not the action was completed (with a default value assigned to the inadequate parameter). The `parameter_number` member identifies the rank of the invalid parameter.

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);

};
```

H.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.

IDL description:

```
interface      RtGenericContent: RtGenericContentOrPresentableSocket, RtComponent {};
```

H.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.

IDL description:

```
interface  GenericPresentableSocket: RtGenericContentOrPresentableSocket, Socket {};
```

H.1.29 `RtContentOrPresentableSocket` object

This subclause defines the operations of the `RtContentOrPresentableSocket` object.

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 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 or not the action was completed (with a default value assigned to the inadequate parameter). The `parameter_number` member identifies the rank of the invalid parameter.

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 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 or not the action was completed (with a default value assigned to the inadequate parameter). The `parameter_number` member identifies the rank of the invalid parameter.

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);

};
```

H.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.

IDL description:

```
interface  RtContent: RtContentOrPresentableSocket, RtGenericContent {};
```

H.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.

IDL description:

```
interface PresentableSocket: RtContentOrPresentableSocket, GenericPresentableSocket
{};
```

H.1.32 `RtMultiplexedContentOrPresentableSocket` object

This subclause defines the operations of the `RtMultiplexedContentOrPresentableSocket` object.

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 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 or not the action was completed (with a default value assigned to the inadequate parameter). The `parameter_number` member identifies the rank of the invalid parameter.

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 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.

IDL description:

```
interface  RtMultiplexedContentOrPresentableSocket{

    void
        setStreamChoice(
            in StreamIdentifier
                stream_identifier)
    raises(InvalidTarget, InvalidParameter);

    StreamValue
        getStreamChosen()
    raises(InvalidTarget);
};
```

H.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.

IDL description:

```
interface  RtMultiplexedContent: RtMultiplexedContentOrPresentableSocket,
RtGenericContent {};
```

H.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.

IDL description:

```
interface  MultiplexedPresentableSocket: RtMultiplexedContentOrPresentableSocket,
GenericPresentableSocket {};
```

H.1.35 `Channel` object

This subclause defines the operations of the `Channel` object. The object inherits from the `Entity` object.

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.

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.

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 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.

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 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.

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 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.

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.

kill operation

Synopsis:

Interface:	`Channel`
Operation:	`kill`
Result:	`void`

Description:

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

setPerceptability operation

Synopsis:

Interface:	`Channel`	
Operation:	`setPerceptability`	
Result:	`void`	
In:	`ChannelPerceptabilityValue`	`channel_perceptability`
Exception:	`InvalidTarget`	

Description:

This operation enables to turn on or off a channel 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 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.

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 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.

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.

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);

};
```

H.1.36 Parameter definition

This 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
// Interface: RtComponentOrSocket Operation: setCurrentTemporalPosition
// 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: 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_PolicyADJUSTMENT_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;
};
```

H.1.37 Exceptions

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.

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 or not the action was completed (with a default value assigned to the inadequate parameter). The parameter_number member identifies the rank of the invalid parameter.

NotBound exception

Description:

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

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.

IDL definition:

```
exception InvalidTarget {
    unsigned short period;
};

exception InvalidParameter {
    enum CompletionStatus { YES, NO};
    ConpletionStatus completion_status;
    unsigned short period;
};

exception AlreadyBound {
    EntityIdentifier entity_identifier;
};

exception NotBound {};
```

History

Document history			
May 1996	Public Enquiry	PE 106:	1996-05-20 to 1996-09-13