INTERNATIONAL STANDARD

ISO/IEC 13522-5

First edition
1997-04-15

Information technology — Coding of multimedia and hypermedia information —

Part 5:
Support for base-level interactive applications

Technologies de l'information — Codage de l'information multimédia et hypermédia —

Partie 5: Support pour applications interactives de niveau fondamental

Reference number
ISO/IEC 13522-5:1997(E)

Contents

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialised system for worldwide standardisation. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organisation to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organisations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting vote.

International Standard ISO/IEC 13522-5 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

ISO/IEC 13522 consists of the following parts, under the general title *Information technology – Coding of multimedia and hypermedia information*:

– *Part 1: MHEG object representation – Base notation (ASN.1)*

– *Part 3: MHEG script interchange representation*

– *Part 4: MHEG registration procedure*

– *Part 5: Support for base-level interactive applications*

– *Part 6: Support for enhanced interactive applications*

Annexes A and B form an integral part of this part of ISO/IEC 13522. Annexes C and D are for information only.

Information technology — Coding of multimedia and hypermedia information —

Part 5:

Support for base-level interactive applications

1 Scope

This part of ISO/IEC 13522 specifies semantics and final-form interchange syntax for MHEG-5 objects, based on concepts defined in ISO/IEC 13522-1. These objects are intended for use in the domain of simple client/server interactive multimedia applications, e.g. (Near) Video on Demand applications, navigation and browsing applications.

1.1 Specificity of the scope

Since it is expected that this part of ISO/IEC 13522 be used for interoperability of applications across platforms, the scope focuses on a specific and precise definition of MHEG-5 classes. This part of ISO/IEC 13522 recognises the semantics implied by the specification of the MHEG-5 objects and by interpretation of MHEG-5 behaviours within the using system.

1.2 Issues outside the scope of this part of ISO/IEC 13522

The scope excludes any standardisation of models, services, systems, protocols or applications that are likely to make use of MHEG-5 objects.

The coded representation of content data is not in the scope of this part of ISO/IEC 13522.

2 Normative References

The following standards contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 13522. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this part of ISO/IEC 13522 are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of ISO and IEC maintain registers of currently valid International Standards.

— ISO/IEC 646:1991: *Information technology – ISO 7-bit coded character set for information interchange.*

— ISO/IEC 8824-1:1995 | ITU-T Recommendation X.680 (1994): *Information technology -- Abstract Syntax Notation One (ASN.1): Specification of basic notation.*

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

— RFC 1521:1993: *MIME (Multipurpose Internet Mail Extensions) Part one: Mechanisms for specifying and describing the format of Internet message bodies.*

3 Terms and definitions

For the purposes of this part of ISO/IEC 13522, the following definitions apply:

3.1 abstract class
Class that is never instantiated into an interchangeable MHEG-5 object
NOTE - An abstract class defines attributes, behaviours and semantics of actions that may be exchanged as parts of any MHEG-5 object of the concrete subclasses of this abstract class.

3.2 action
Set of elementary actions

3.3 active
State of any MHEG-5 object when the Activation behaviour has been completed successfully for this object
An active object has its *RunningStatus* set to *True*.

3.4 application domain
Specific domain of applications making use of this part of ISO/IEC 13522 and providing additional tools and values in order to create a practical instance of an MHEG-5 environment
NOTE - More information on application domains is provided in Annex D of this part of ISO/IEC 13522.

3.5 application scope
Common scope of all MHEG-5 objects (scenes and ingredients) accessed from an MHEG-5 application

3.6 attribute
Named and typed value attached to a class

3.7 available
State of any MHEG-5 object when the Preparation behaviour has been completed successfully for this object
An available object has its *AvailabilityStatus* set to *True*.

3.8 base class
MHEG-5 class which defines some attributes, behaviours and semantics of actions that are shared by a given MHEG-5 class

3.9 concrete class
Class of any MHEG-5 object that may be encoded and interchanged according to the specifications provided in Annex A or Annex B of this part of ISO/IEC 13522

3.10 elementary action
Abstract representation of a message that may be sent to an object
NOTE - This part of ISO/IEC 13522 defines the semantics of available elementary actions for each MHEG-5 class. Note that the MHEG-5 class Action (with a capital A) has a different meaning described in clause 49 of this part of ISO/IEC 13522.

3.11 event
Abstract representation of an occurrence of a special meaning for any MHEG-5 object
NOTE - Events are used to trigger Link conditions and bring out the execution of sequences of elementary actions.

3.12 exchanged attribute
Attribute that is part of the interchangeable byte-code representation of an MHEG-5 object, and transmitted with that object

3.13 inactive
State of any MHEG-5 object when the Deactivation behaviour has been completed successfully or when no Activation behaviour has been applied successfully on this object
An inactive MHEG-5 object has its *RunningStatus* set to *False*.

3.14 inherited attribute
Attribute that is defined in a base class of the class of the MHEG-5 object

3.15 interchangeable representation
Octet string, which contains the encoded exchanged attributes of this MHEG-5 object complying with the ASN.1 syntax and encoding provided in Annex A of this part of ISO/IEC 13522, or, when textual representation is preferred, with Annex B of this part of ISO/IEC 13522

3.16 internal attribute
Abstract data structure, never byte-code encoded or exchanged, that is used to define semantics of internal behaviours or actions for any MHEG-5 object
NOTE - Any MHEG-5 engine might consider that an internal attribute is part of the internal representation of the MHEG-5 object; however, this is not mandatory. What is mandatory is to implement the functionality described by these internal attributes.

3.17 internal behaviour
Abstract function that defines the semantics of MHEG-5 elementary actions for any MHEG-5 class
NOTE - An internal behaviour of a class is most of the time overridden by internal behaviours of subclasses of this class. An MHEG-5 engine might consider that an internal behaviour of a class is a private method of this class, however this is not mandatory.

3.18 MHEG-5 application
Set of scenes and control information that allows the user to navigate between scenes
NOTE - the MHEG-5 class Application (with a capital A) has a different, more specific meaning that is given in clause 10 of this part of ISO/IEC 13522.

3.19 MHEG-5 class
Abstract definition of exchanged and internal attributes of parts of interchangeable multimedia / hypermedia objects as well as definition of the semantics of internal behaviours and the effect of MHEG-5 actions for these objects

3.20 MHEG-5 engine
Process or set of processes that interpret MHEG-5 objects encoded according to the encoding specifications defined in Annex A or in Annex B of this part of ISO/IEC 13522

3.21 MHEG-5 object
Instance of any MHEG-5 class
NOTE - An MHEG-5 object is not a physical object, but rather an abstraction that may have many representations of different types. Various software services handle such representations.

3.22 MHEG-5 scene
Structure that co-ordinates the presentation (visual and audible) of MHEG-5 objects

3.23 mix-in class
Abstract class that does not inherit from the Root class
Examples: *Interactible* class, *TokenManager* class.

3.24 non-available
State of any MHEG-5 object when the Destruction behaviour has been completed successfully or when no Preparation behaviour has been applied successfully on this object
A non-available object has its *AvailabilityStatus* set to *False.* Even if an MHEG-5 object does not exist in the MHEG-5 engine, its *AvailabilityStatus* exists and is set to *False.*

3.25 subclass
Any MHEG-5 class that shares the same attributes, behaviours and semantics of actions as another MHEG-5 class.

4 Conformance

This clause specifies conformance requirements for MHEG-5 engines and for MHEG-5 applications.

4.1 Conformance of MHEG-5 objects

Any MHEG-5 object shall have an octet representation. For interchange purposes, the octet representation shall be compliant with the ASN.1 syntax and encoding defined in Annex A, or with the textual notation grammar defined in Annex B. The application domain shall choose which representation to use: that of Annex A or that of Annex B; and that representation shall then be used exclusively throughout the application domain.

The attributes of any MHEG-5 object shall meet all requirements defined in the relevant subclauses of this part of ISO/IEC 13522.

4.2 Conformance of MHEG-5 engines

Conformance of MHEG-5 engines can only be measured with regard to a complete application domain definition. To fully specify conformance, an application domain shall define, in addition to the interchange representation, the following:

1. a set of classes from the list of all classes of this part of ISO/IEC 13522, as prescribed in 4.2.1
2. a set of features from the list in 4.2.2
3. additional concrete choices as listed in 4.2.3.

NOTE - Refer to Annex D of this part of ISO/IEC 13522 for an example of complete definition of an application domain.

4.2.1 Conformance to the acceptance of a set of Classes and Elementary Actions

Conformance to the acceptance of a set of Classes and Elementary Actions is defined as follows.

Any MHEG-5 engine is required to implement at least the following minimum set of classes:

- Application Class

  All attributes, events and internal behaviours shall be implemented.
- Scene Class

  All attributes, events and internal behaviours shall be implemented.
- Link Class

  All attributes, events and internal behaviours shall be implemented.
- Action Class

  All attributes, events and internal behaviours shall be implemented.

All application domains shall define compliance to a set of classes containing at least the minimum set above. An application domain may specify a larger set of classes and elementary actions for compliance; in any case, the application domain shall clearly list the classes and elementary actions supported.

When additional classes are implemented in any MHEG-5 engine, the engine shall implement all of their attributes, events, internal behaviours and elementary actions as defined in this part of ISO/IEC 13522, with the possible exception of optional features listed in 4.2.2. Concerning the Action class, the engine shall implement all effects of MHEG-5 elementary actions corresponding to the specified set of classes . It is the role of each application domain to choose and well define a set of classes that is required for that specific application domain.

If a class is not handled by an MHEG-5 engine, and an object of this class is sent to the MHEG-5 engine, this causes an error that is handled by the default error handling defined in subclause 53.7.

4.2.2 Conformance to a set of engine functionality

Conformance to a set of engine functionality is defined as follows.

Any MHEG-5 engine shall provide all normative mechanisms defined in clauses 51 to 54.

Any MHEG-5 engine shall implement all effects of MHEG-5 actions and the internal behaviours of MHEG-5 classes included in the definition of their application domain, except for the following optional features:

- Ancillary connections (corresponding to OpenConnection and CloseConnection actions),
- Caching, (corresponding to caching of MHEG-5 objects and content data of Ingredient objects),
- Cloning, (corresponding to the Clone action defined in Ingredient class).
- Free-moving cursor,
- Bitmap and Video scaling, (corresponding to the ScaleBitmap and ScaleVideo actions of the Bitmap and Video classes).
- Stacking of Applications, (corresponding to the Spawn action of the Application class),
- Trick mode, (corresponding to the SetSpeed action of the Stream class),

An application domain shall clearly define a list of which ones of the above features are mandatory or optional for conformance to the application domain.

4.2.3 Additional requirements for conformance specification

In addition to the two items above, the following tables shall be specified by a given application domain to fully define conformance.

NOTE - For each one of these table, a concrete example is given in Annex D (informative), thus defining an example of application domain.

- Content Data Encoding

  The application domain shall specify which type of content data is supported and which type of encoding is supported. The following two tables shall be filled for each application domain.

Attribute	Permissible values
FontAttributes	
FontName	
AbsoluteColour	
CharacterSet	
TransitionEffect	

Encoding table:

Type of content	Content Encoding	Hook Values (Integer)
Font encoding format		
Palette encoding format		
Bitmap encoding format		
Text encoding format		
EntryField encoding format		
HyperText encoding format		
Stream encoding format		
LineArt encoding format		
CursorShape encoding format		
InterchangedProgram encoding format		
AbsoluteColour encoding format		

- UserInput registers

  In order to have a working instantiation standard, the application domain shall specify one or more InputEventRegisters. Each register has a number, which is exchanged as one of the parameters of a Scene object and a content (which is not exchanged) consisting of a set of numbers (representing UserInputEventTags) and a name. The name/number pairs bind a specific UserInputEventTag to a logical input event. It is the task of the engine implementer to bind the logical input event to one or more physical input events.

  The following table shall be filled for each application domain.

Register #	UserInputEventTag	Semantics	Comment
(Integer)	...	...	...

- EngineEvent

  An MHEG-5 application domain may specify a set of numbers associated with EngineEvent to distinguish between the various external events that lead to the generation of the EngineEvent. In such case, each of these specific EngineEvent shall be mapped to a corresponding Integer in table such as the one below. Values not reserved by the application domain are free for use by the application programmer.

EngineEvent	EventTag
...	*(Integer)*

- GetEngineSupport

  Application domains may define, in addition to the strings mentioned in this part of ISO/IEC 13522, other permissible strings for the GetEngineSupport action. In such case, the application domain shall clearly list these additional strings.

- Semantic constraints on the MHEG-5 applications

  For each feature defined by a string for the GetEngineSupport action, an application domain may choose to constrain its applications in some way. In such cases a table of constraints shall be provided.

Feature	Constraint
...	...

- Protocol mapping and external interaction

  Finally there are the different actions that have an effect that is external to the runtime. These are functions that retrieve objects, manipulate streams and call external function. For interoperability, these actions shall have a common underlying external effect. This usually implies that there is a consistent mapping of these actions on the underlying external communications functions for users of MHEG-5 application domains. The following mappings shall be provided by the application domain:

MHEG-5 entity	Mapping needed.	Semantics of MHEG-5 structures that needs specification:

OpenConnection, CloseConnection	Mapping to connection management (and possibly session management) protocols in the application domain	• In OpenConnection:
		  • Protocol
		  • Address

RemoteProgram objects	Mapping to RemoteProgram call protocol in the application domain	• In Call and Fork:
		  • Name
		  • Parameters
		  • ProgramConnectionTag

Application name space	Mapping to name space of the application domain	• ObjectReference • ContentReference
Application name space in case a TransitionTo action uses the ConnectionTag parameter	Mapping to the name space of the application domain	• ObjectReference • ContentReference
Persistent storage name space	Mapping to the name space of the persistent storage	• In StorePersistent and ReadPersistent: • InFileName, OutFileName

Stream actions	Mapping to the stream interface of the application domain	• In Stream
		  • Speed
		  • CounterPosition

Stream events	Mapping to stream states and stream events in the application domain	• In Stream
		  • StreamPlaying, StreamStopped (mapping to application-domain stream state machine)
		  • CounterPosition
		  • StreamEventTag

5 Overview of the MHEG-5 Classes

This part of ISO/IEC 13522 was developed to support the distribution of interactive multimedia applications in a client/server architecture[1] across platforms of different types and brands. This part of ISO/IEC 13522 defines a final-form representation for application interchange. The applications consist mainly of declarative code, but provisions for calling procedural code have been made. MHEG-5 applications need only be authored once and then runs on any platform that is compliant to this part of ISO/IEC 13522. The developed applications would reside on the server, and as portions of the application are needed, they will be downloaded to the client. In a broadcast environment, this download mechanism could rely, for instance, on cyclic rebroadcasting of all portions of the application. It is the responsibility of the client to have a runtime that interprets the application parts, presents the application to the user, and handles the local interaction with the user.

Any MHEG-5 application is made up of scenes and objects that are common to all scenes. A scene contains a group of objects used to present information (graphics, sound, video, etc.) along with localised behaviour based on events firing (e.g. the Left button being pushed activates a sound). At most one scene is active at any one time. Navigation in an application is done by making transitions between scenes.

The interactive system has the ability to display visual objects in a rectangular coordinate system with a fixed size, and to play audible objects. User input devices (e.g. remote control, game controller, etc.) may be used with the runtime to allow interaction with the applications.

The figures in this informative clause present the class diagram of the object classes defined by this part of ISO/IEC 13522. Their meaning is explained in Figure 1.

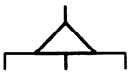
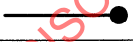
Link	The boxes depict an MHEG-5 class. Class names in bold depict a concrete class.
Visible	Class names in normal style depict an abstract class.
	Triangles depict inheritance relationships.
	Diamonds depict composition relationships.
	Dark circles depict a zero-to-more relationship.

Figure 1 - Legend of Class diagrams

The Figure 2 presents an overview of the top layer in the MHEG-5 class hierarchy. The next part of this clause introduces the concepts defined by this part of ISO/IEC 13522 by explaining the classes shown in this Figure and their subclasses.

[1] The server can be a *virtual* server, such as the collection of a number of broadcast channels on a broadcast network.

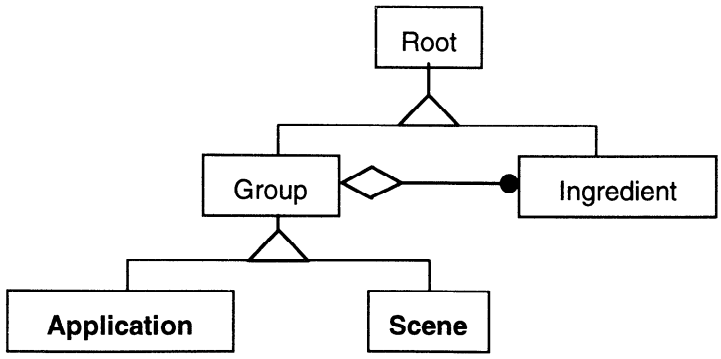


Figure 2 - Diagram of the top of the class hierarchy

5.1 Root

This is the abstract base class for all other MHEG-5 classes[2]. Its main functionality is to provide semantics for generic MHEG-5 behaviour (activation, deactivation, preparation, destruction), and to provide a mechanism for object identification.

5.2 Group

This is an abstract base class for the classes Application and Scene. Its main functionality is that of allowing the grouping of objects of other classes for exchange between the MHEG-5 engine and other entities (similar to a «set» in standard object-oriented terminology). The objects that are grouped by this class are objects of the class Ingredient. Each Ingredient is always contained in exactly one Group. Objects within a Group may be referenced from objects in other Groups under certain conditions (see below).

5.3 Application

Objects of the Application class group objects of the Ingredient class. The Application class also has the semantic constraint that only one Application object may be active[3] at once, and that no other objects may be active unless an Application object is active.

An idle MHEG-5 engine starts an application by preparing and activating the corresponding Application object. When the Application object becomes active, it automatically runs an OnStartUp action, which can be used to run the first Scene object of the application. Since (exactly) one Application object is active whenever another object is active, the Ingredients contained in the Application object are visible and available to other objects that are active simultaneously. More specifically, any Ingredients contained in an Application object are available to the active Scene. This can be used to describe application-wide behaviour.

5.4 Scene

Objects of the Scene class group objects of the Ingredient class. The purpose of the Scene class is to allow spatially and/or temporally co-ordinated presentation. Only one Scene object may be active at a time within an MHEG-5 engine. A Scene object must be active in order for any ingredient (be it contained in a Scene or in an Application object) to be displayed.

The Scene class provides the special action TransitionTo, which makes it possible to perform a graphical transition between two scenes. Objects contained in a scene can only be displayed when that scene is active. Object that need to be displayed across several scenes (e.g., to have uninterrupted presentation over a scene transition must be contained in an application object. Finally, the Scene class provides information about the coordinate system to be used for visual presentation.

---

[2] With the exception of the abstract mix-in classes and of the Action class.

[3] For the definition of the term active, see clause 3.

5.5 Ingredient

The Ingredient class is an abstract base class for the classes Link, Program, Palette, Font, CursorShape, Variable and Presentable. The subclasses of the Ingredient class are presented in Figure 3. The main functionality of the Ingredient class is to specify the generic behaviour of objects that can be part of a Scene or an Application object.

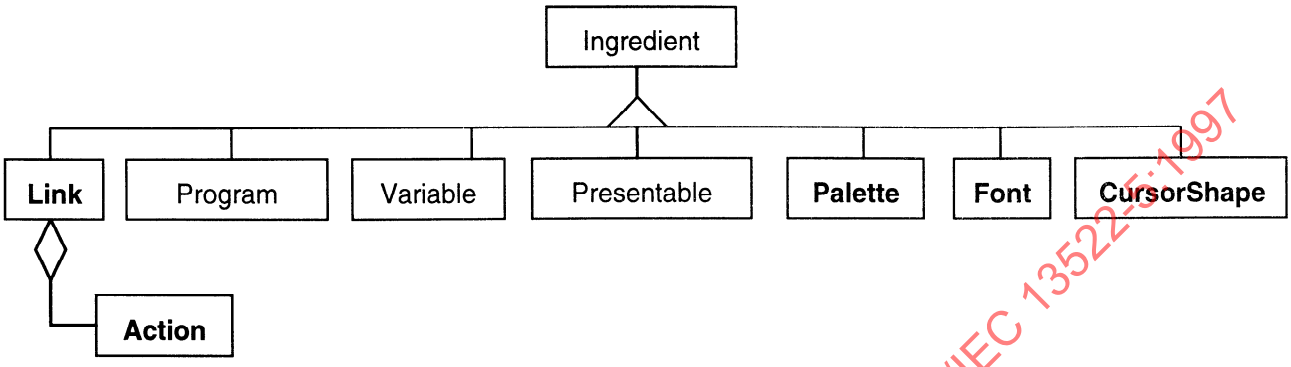


Figure 3 - Diagram of subclasses of the Ingredient class

5.6 Link

Link objects are used to express the behaviour of MHEG-5 applications. A Link object consists of a condition and an Action object. When the condition part evaluates to True, the Link is said to «fire»; this leads to the running of the associated Action object. The condition contains three parts: an event code (identifying which event has to be reacted on), a reference to the object from which the event should emanate, and a value that specifies the required value of the event parameter. In other words, a Link fires only if it is active and only if the right event is generated by the right object and contains the right event parameter. Active Links shall be part of either an active Scene or an active Application.

5.7 Action

An Action object has the functionality of executing, in synchronous sequence, a series of «elementary actions» as the result of a Link firing. An elementary action consists of the object to which the action is to be «targeted» and a list of values representing the parameters of the action. In fact, targeting an elementary action to an object corresponds to calling a method of an object in any ordinary object-oriented programming language. All elementary actions available are listed in clause 49.

The Action class does not inherit from any other MHEG-5 classes. Specifically, it does not inherit from Root, which means that Action objects cannot be addressed as individual entities.

5.8 Program

The Program class provides the functionality of calling a piece of procedural code from within the MHEG-5 context and exchanging parameters with it.

The Program class has the following three subclasses corresponding to the three types of procedural calls that can be made:

- ResidentProgram

  Procedural call to a piece of code that is specific to the device on which the MHEG-5 engine is running. It can be used, for instance, to call device-specific runtime libraries.

- RemoteProgram

Procedural call to a piece of code that is located on a device different than the one where the MHEG-5 engine is running. It can be used, for instance, to implement a remote program call, where the actual body of the program is located at the server in a client-server system.

- InterchangedProgram

  Procedural call to a piece of code which is exchanged as a part of an MHEG-5 object. The purpose of this class is to provide functionality needed for exchanging pieces of procedural code, and for calling those pieces of code.

5.9 Palette, Font, and CursorShape

The Palette class provides the possibility to encapsulate the encoded representation of a colour look-up table (CLUT). The function of a CLUT is to translate a colour index to a true colour value. A Palette may be used, for instance, with bitmaps to specify the colours in which the bitmap is to be rendered.

Similarly, the Font class allows applications to encapsulate the encoded representation of a font. A Font object, when associated with a Text object, is used to render the text of that object.

The CursorShape class, finally, allows applications to encapsulate the encoded representation of the bitmaps, mask, and other data needed to render a free-moving cursor. The free-moving cursor shape can be set and reset using a method of the Scene class.

For Fonts, Palettes and CursorShapes, the actual representation of the objects is not specified by this part of ISO/IEC 13522. However, an application domain where fonts and/or CLUTs and/or free-moving cursors are necessary features of applications can specify their encoding and semantics.

5.10 Variable

The Variable class provides the possibility to store and retrieve values. The Variable class has five subclasses corresponding to five different types of variables:

- BooleanVariable
- IntegerVariable
- OctetStringVariable
- ObjectRefVariable
- ContentRefVariable

Possible uses of Variables include parameter passing to and from Program calls, storage of the state of other MHEG-5 objects, passing indirect parameter values to actions and address indirection, i.e. as pointers to MHEG-5 objects.

5.11 Presentable

The Presentable class is an abstract base class for the MHEG-5 classes Audio, Visible, TokenGroup, and Stream. The subclasses of the Presentable class are presented in Figure 4.

Objects of the Presentable class represent information that can be directly seen or heard by the user. An important functionality of the Presentable class is to handle the actual encoded representation of the content data. This can be done either by «inclusion» or by «reference». In the former case, the content data is actually transmitted as part of the Presentable object itself; in the latter case, the Presentable object merely provides an external reference to the data.

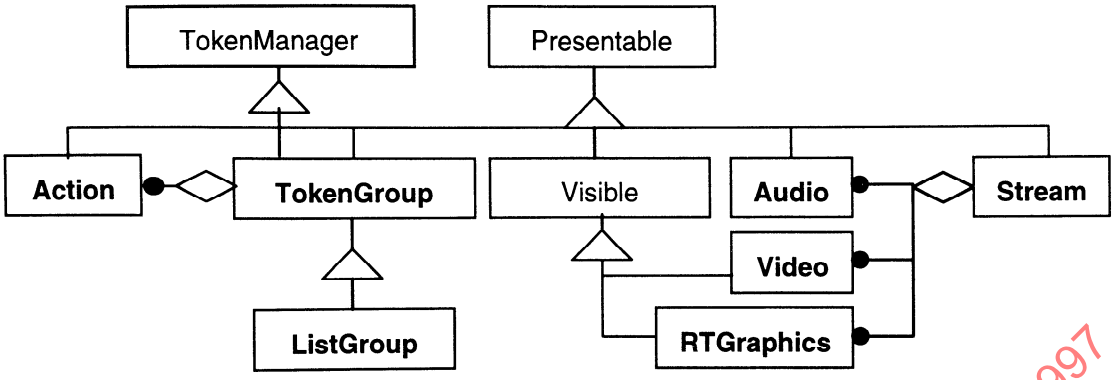


Figure 4 - Diagram of subclasses of the Presentable class

5.12 TokenGroup

The TokenGroup class provides the facility to navigate a logical token among a set of Visible objects. This structure can be used, for example, to manage the navigation of a «focus» among a set of buttons or other elements of a Scene. Some sets of actions may also be attached to the objects and executed on demand on the visible object that has the token. The latter feature provides a compact way of expressing behaviour when an element of the group gets or loses the focus.

5.13 ListGroup

The ListGroup class completes the TokenGroup class, providing functionality for selecting objects in a long list. It is best suited to implement a menu for selection, a group of checkboxes, a carrousel of bitmaps, a fill-in form, a scrollable list of items, etc. In addition, the ListGroup class provides facilities to dynamically add and remove items to the group.

5.14 Stream

The Stream class defines a multiplex of continuous media for synchronisation in time. Audio, Video and RTGraphics objects might be elementary streams of a Stream multiplex: they are intended to be presented at the same time to the user. This structure can be used, for instance, to present video and audio synchronously and to switch from one audio channel to another. The content data of the Stream object is a reference to a real multiplex containing the elementary streams and some additional data for synchronisation. During the rendering process, the Stream player generates time-based events and marker-based events that might be used by the MHEG-5 application to trigger some Links.

5.15 Audio

The Audio class implements a sequence of audio data that can be used as an elementary stream of a Stream multiplex.

5.16 Interactible

The Interactible class is an abstract mix-in class inherited by the MHEG-5 classes HyperText, EntryField, Slider, and Button. Its main functionality is that of allowing the user to interact with objects of its sub-classes. These interactions enable the user to change the status and/or appearance of the objects, for instance by entering text in an EntryField object. When interaction is taking place, a certain class of events, called the «UserInput events» are not visible to Link objects, since those events are assumed to be used for the user interaction. The interaction can be aborted by targeting a certain action to the Interactible object.

Another functionality of the Interactible class is the ability to generate events associated with free-moving cursors (CursorEnter, CursorLeave).

5.17 Visible

The Visible class implements the functionality associated with rendering pieces of visual material at some location on the display screen. The subclasses of the Visible class are presented in Figure 5.

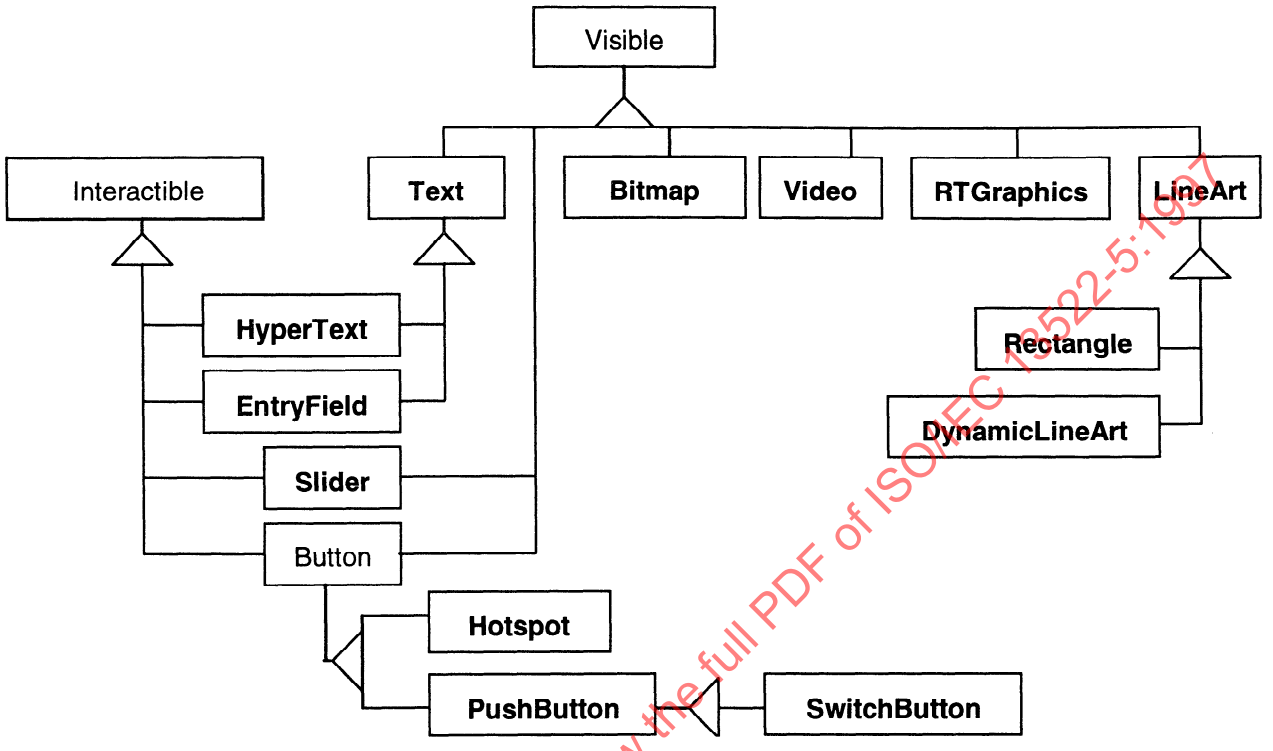


Figure 5 - Diagram of subclasses of the Visible class

The subclasses of the Visible class are described below:

- LineArt

  An object of the LineArt class represents a graphical object with vectorial representation. It can be used, for instance, to present polyline objects, ellipses, bezier curves, etc.

- DynamicLineArt

  An object of the DynamicLineArt class represents a graphical object that can be dynamically changed. It can be used to draw lines or curves that need to be presented on the fly.

- Rectangle
- Bitmap
- Video
- RTGraphics

  An object of the RTGraphics (Real Time Graphics) class represents a stream of graphics objects that are displayed using autonomous placement and synchronisation. The RTGraphics stream can be used in conjunction with video and audio, for instance, to create a subtitling application.

- Text

  Text objects represent text strings. A Text object may be associated with a Font object, which describes the font in which the text should be rendered (in case no font object is given, the default font is assumed). Text has two subclasses, HyperText and EntryField, which implement different types of interactive text.

- Slider

A Slider is an Interactible used to let the user set linearly a position within a certain range (given by a minimum and maximum value).

- Button

The Button class has the two subclasses: PushButton and Hotspot. The PushButton class has one subclass: SwitchButton. Buttons are rectangular areas on the screen with which the user can interact. Each of the three types of buttons has a specific event-generating behaviour attached to it. PushButtons and SwitchButtons are associated with a text item, which represents the text to be displayed in the middle of the button.

6 Structure of this part of ISO/IEC 13522

The following clauses of this part of ISO/IEC 13522 defines the semantics of the MHEG-5 classes. This is done on the basis of an abstract syntax notation used to describe the attributes of the exchanged objects. The semantics of the object classes are described in normative text. Clauses 51 to 54 of this part of ISO/IEC 13522 define some normative mechanisms that the MHEG-5 engine is required to implement. Annex A of this part of ISO/IEC 13522 defines the final-form syntax to be used for object exchange. Annex B of this part of ISO/IEC 13522, defines a textual interchange format which maps one-to-one on the final-form binary exchange format.

7 Notations

The following notations are used in the following clauses to describe the MHEG-5 classes defined by this part of ISO/IEC 13522.

<Name of the Class>

Description	<Short description of the semantics of the class>
Base class	<Name of the base class>
Subclasses	<List of subclasses, if any>
Status	<Abstract class | Concrete class>

7.1 Attributes

This subclause defines inherited, exchanged and internal attributes for the class.

7.1.1 Inherited attributes

Requirements and constraints on attributes inherited from the base classes.

Attribute Name	Defined in	Constraints and Requirements
<Attribute Name>	<Class name>	<Specific constraints for the current subclass>.

7.1.2 Own exchanged attributes

List of exchanged attributes for this class.

<Attribute Name> <Description of the attribute>
<Type of the attribute>
<Default value>

When the attribute is optional, default value is the value that shall be used when the attribute is not encoded. When the attribute is mandatory, default value is a hint on the most usual value to use.

7.1.3 Own internal attributes

List of internal attributes for this class.

<Attribute Name> <Description of the attribute>
<Initial value>

7.2 Events

These are the events that can be generated from objects of this class. They are used to express Link conditions that shall be checked when such an event is generated.

<Event Name> <Description of the event>
<Context of occurrence of the event>
<Description of the event data associated with the event, if any>

7.3 Internal behaviours

The following internal behaviours are defined for most MHEG-5 classes:

- *Preparation*
- *Destruction*
- *Activation*
- *Deactivation*

In addition, the *Interaction* internal behaviour is defined for some MHEG-5 classes. These behaviours correspond to semantic requirements for operations internal to the MHEG-5 engine, similar to the notion of «private» methods of a class in object-oriented terminology. They are not encoded nor transmitted within an Application description.

When a behaviour is not described for an MHEG-5 class, the semantics of the behaviour for its base class applies.

<Behaviour name> <Semantics of the behaviour within the context of the current class.>

7.4 Effect of MHEG-5 actions

Define the semantics and syntax of MHEG-5 actions that may be targeted to the current MHEG-5 class.

<Action Name> <Semantics of the action within the context of the current class.>

7.5 Formal description

Description of an encoded MHEG-5 object of the current class in Extended Bacchus-Naur Form (EBNF):

```
Class Name          -->  First part,
                         Second part
First part          -->  Subpart | Alternative
Second part         -->  Terminal (constant or type)
```

8 Root Class

Description	Root class of all MHEG-5 classes.
Base class	None
Subclasses	Group, Ingredient
Status	Abstract class

8.1 Attributes

This subclause defines inherited, exchanged and internal attributes for this class.

8.1.1 Inherited attributes

This class has no inherited attributes.

8.1.2 Own exchanged attributes

This class defines the following additional exchanged attributes:

ObjectIdentifier — This is a mandatory data structure, which consists of the following parts:

- Optional GroupIdentifier.

  Unique identifier of a group of MHEG-5 objects.

  - Optional OctetString.
  - Default value: The GroupIdentifier of the Group within which the object is encoded.

  The actual structure of this parameter is not defined by this part of ISO/IEC 13522. However, the application domain shall define such a structure. See clause 51.

- ObjectNumber.

  Unique identifier of any MHEG-5 object within a group.

  - Integer.

8.1.3 Own internal MHEG-5 attributes

This class defines the following internal attributes:

AvailabilityStatus — State of availability of the object.

When the *AvailabilityStatus* of the object is True, the object is available, this means that the *Preparation* behaviour of the object has ended successfully.

When the *AvailabilityStatus* of the object is False, the object is non-available, this means that the *Preparation* behaviour has not ended successfully or has not been invoked, or that the *Destruction* behaviour has been applied successfully.

- Boolean value.
- Default value: False.

RunningStatus — State of activity of the object.

When the *RunningStatus* of the object is True, the object is active, this means that the *Activation* behaviour of the object has ended successfully.

When the *RunningStatus* of the object is False, the object is inactive, this means that the *Activation* behaviour has not successfully ended or has not been invoked, or that the *Deactivation* behaviour has been applied successfully.

- Boolean value.
- Default value: False.

8.2 Events

This class defines the following events:

IsAvailable This event is generated when the *AvailabilityStatus* attribute changes from False to True as a result of the *Preparation* behaviour ending successfully for the object.

NOTE - The purpose of this event is to indicate to the MHEG-5 engine that the object is available and can be activated.

- No associated data.

ContentAvailable This event is generated when the object and its content are available in an optimal state to the MHEG-5 engine. Its purpose is to indicate to the MHEG-5 engine that the activation behaviour can happen in a timely manner. This event is generated asynchronously with the *Preparation* behaviour for the object.

NOTE - Each MHEG-5 engine may choose to give a different meaning to that degree of content availability, this part of ISO/IEC 13522 does not specify any meaning or time requirement in this matter.

- No associated data.

IsDeleted This event is generated when the *AvailabilityStatus* attribute changes from True to False as a result of the *Destruction* behaviour ending successfully for the object.

- No associated data.

IsRunning This event is generated when the *RunningStatus* attribute changes from False to True as a result of the *Activation* behaviour ending successfully for the object.

- No associated data.

IsStopped This event is generated when the *RunningStatus* attribute changes from True to False as a result of the *Deactivation* behaviour ending successfully for the object.

- No associated data.

8.3 Internal behaviours

This class defines the following internal behaviours:

Preparation This behaviour has the basic semantics of allocating all requested resources in order to handle or to present this object.

Apply the following sequence of actions:

1. If the *AvailabilityStatus* attribute of the object is True, abort the behaviour. Otherwise:

2. Retrieve the object from an entity outside the engine.
3. Set each internal attribute of the object to its initial value.
4. Set the *AvailabilityStatus* attribute to True.
5. Generate an *IsAvailable* event.

The above steps are executed *synchronously*. The following step is asynchronous.

6. Generate a *ContentAvailable* event.

Destruction

This behaviour has the basic semantics of asking the MHEG-5 engine to delete the object.

Apply the following sequence of actions:

1. If the *AvailabilityStatus* attribute of the object is False, abort the behaviour. Otherwise:
2. If the *RunningStatus* attribute of the object is True,
   a) apply the *Deactivation* behaviour.
   b) wait for an *IsStopped* event from the object.

   This shall all be done synchronously.
3. If the *RunningStatus* attribute of the object is False, execute the following actions synchronously.
4. If the *GroupCachePriority* attribute of the object itself or of the group this object belongs to is set to 0, the MHEG-5 engine shall free all resources allocated to the object.
   Note that *GroupCachePriority* is defined in Group class.
5. If the *GroupCachePriority* attribute of the object itself or of the group this object belongs to is different from 0, the MHEG-5 engine may decide to either actually free all resources allocated to the object or to cache it.
6. Generate an *IsDeleted* event.

   Note that the *IsDeleted* event shall be generated whether the resources mentioned above were actually freed or not; the object is deleted in the sense defined by this part of ISO/IEC 13522, even if some associated resources are not.

Activation

This behaviour has the basic semantics of immediately making this object active.

Apply the following sequence of actions:

1. If the *RunningStatus* attribute of the object is True, abort the behaviour. Otherwise:
2. If the *AvailabilityStatus* attribute of the object is False,
   a) apply the *Preparation* behaviour to the object.
   b) wait for an *IsAvailable* event from the object.

   These steps are executed synchronously, meaning that the engine will not perform other actions until the *Preparation* behaviour has ended.

NOTES

1 The effect of the *Activation* behaviour (e.g., the display of a bitmap) will continue even after the behaviour itself has returned.

2 The generation of an *IsRunning* event and the modification of the *RunningStatus* internal attribute are parts of the *Activation* behaviour of subclasses of the Root class.

Deactivation This behaviour has the basic semantics of notifying the MHEG-5 engine to deactivate this object immediately.

Apply the following sequence of actions:

1. If the *RunningStatus* attribute of the object is False, abort the behaviour. Otherwise:
2. Set the *RunningStatus* attribute of the object to False.
3. Generate an *IsStopped* event.

8.4 Effect of MHEG-5 actions

This class defines the following applicable MHEG-5 actions:

GetAvailabilityStatus (*AvailabilityStatus-Var*) Set the Variable referenced by *AvailabilityStatusVar* to the value of the *AvailabilityStatus* attribute.

NOTE - A GetAvailabilityStatus action targeted to an object inexistant in the MHEG-5 engine is not an error; the result is False.

Provisions of use:

- *AvailabilityStatusVar* shall refer to an active BooleanVariable object.

Syntax description:

```
GetAvailabilityStatus      -->   Target,
                                 AvailabilityStatusVar
Target                     -->   GenericObjectReference
AvailabilityStatusVar      -->   ObjectReference
```

GetRunningStatus (*RunningStatusVar*) Set the Variable referenced by *RunningStatusVar* to the value of the *RunningStatus* attribute.

Provisions of use:

- The *Target* object shall be available.
- *RunningStatusVar* shall refer to an active BooleanVariable object.

Syntax description:

```
GetRunningStatus           -->   Target,
                                 RunningStatusVar
Target                     -->   GenericObjectReference
RunningStatusVar           -->   ObjectReference
```

8.5 Formal description

```
Root Class                 -->   ObjectIdentifier
ObjectIdentifier           -->   ObjectReference
```

9 Group Class

Description	Defines the structure and behaviour of objects used as composition of Ingredients.
Base class	Root
Subclasses	Scene, Application
Status	Abstract class

9.1 Attributes

This subclause defines inherited, exchanged and internal attributes for this class.

9.1.1 Inherited attributes

This class has all the attributes of its base class, with the following constraints:

Attribute Name	Defined in	Constraints and Requirements
ObjectIdentifier	Root	This attribute is mandatory for this class. The Group Identifier part of this attribute is mandatory and shall be unique within the name space of the application domain. The Object Number part of this attribute shall be set to zero.

9.1.2 Own exchanged attributes

This class defines the following additional exchanged attributes:

StandardIdentifier — This is an optional sequence of two Integers. When encoded,

- the first Integer shall always be set to 2, signifying «Joint ISO ITU».
- the second Integer shall always be set to 19, signifying MHEG.

StandardVersion — This is a single optional Integer. It defines the version of this part of ISO/IEC 13522 to which the Group objects, and all its Items, conform.

- Optional Integer. If encoded, its value shall be 1.
- Default value: 1.

ObjectInformation — This is an optional OctetString. When encoded, it shall hold information about the objects encoded in this Group. Such information may comprise information about object Name, Owner, Version, Date, Keywords, Copyright, License, and Comments.

OnStartUp — Set of elementary actions to run during the *Activation* behaviour for the Group.

- Optional inclusion of an Action object.
- Default value: None.

OnCloseDown — Set of elementary actions to run at the beginning of the *Deactivation* behaviour for the Group.

- Optional inclusion of an Action object.
- Default value: None.

OriginalGroup-CachePriority — Hint to the MHEG-5 engine regarding the relevance of caching this Group of Ingredients when it is destroyed.

Value of the *GroupCachePriority* when the Group is prepared.

- Optional Integer within the range [0, 255].
- Default value: 127.
- Specific value: 0 means cache is not allowed for this Group and its Ingredients.

NOTE - As specified in 4.2.2, caching of any kind is an optional feature of an MHEG-5 engine.

Items — Set of Ingredient objects that belong to the Group. When the Group contains no Ingredients, this attribute shall not be encoded. When this attribute is encoded it shall contain at least one Ingredient.

- Optional Attribute.
- Sequence of inclusions of Ingredient objects.
- Default value: None.

9.1.3 Own internal attributes

This class defines the following internal attributes:

GroupCachePriority — Hint to the MHEG-5 engine regarding the relevance of caching this Group of Ingredients when it is destroyed.

The *GroupCachePriority* may be compared with the *GroupCachePriority* of other Group objects to determine which of a number of groups has the highest likelihood of being required again by the application, once it has been destroyed. A higher value indicates a higher level of priority. It is the responsibility of the application designer to keep these numbers to a consistent range. The MHEG-5 engine is recommended to cache objects with higher priority in preference to objects with lower priority.

- Optional Integer within the range [0, 255].
- Initial value: Value of the OriginalGroupCachePriority attribute.
- Specific value: 0 means cache is not allowed for this Group and its Ingredients.

9.2 Events

This class has the same events as its base class, with identical semantics.

9.3 Internal behaviours

The following internal behaviours semantics have changed from this object's base class:

Preparation

1. Apply the *Preparation* behaviour to all Ingredients of the Group that have the InitiallyActive attribute set to True and to all Programs of the Group that have the InitiallyAvailable attribute set to True in the order that they are listed in the Items attribute.
2. Apply the *Preparation* behaviour as inherited from the base class.

Destruction

1. Apply the *Destruction* behaviour to all Ingredients of the Group in the reverse order that they are listed in the Items attribute.
2. Apply the *Destruction* behaviour as inherited from the base class.

Activation

1. Apply the *Activation* behaviour as inherited from the base class.
2. Run the action contained in the OnStartUp attribute.
3. Apply the *Activation* behaviour to all Ingredients of the Group that have the InitiallyActive attribute set to True, in the order they are listed in the *Items* attribute.
4. Set the *RunningStatus* attribute of the Group object to True.
5. Generate an *IsRunning* event.

Deactivation

If group is not active, ignore behaviour. If group is active, do the following three steps:

1. Run the action contained in the OnCloseDown attribute.
2. Apply the *Deactivation* behaviour to all active Ingredients of the Group, in the reverse order they are listed in the *Items* attribute.
3. Apply the *Deactivation* behaviour as inherited from the base class.

9.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base class, with identical semantics. In addition, the following applicable MHEG-5 actions are defined:

SetCachePriority (*NewCachePriority*)

Set the *GroupCachePriority* attribute to *NewCachePriority*.

Provisions of use:

- The *Target* object shall be available.
- *NewCachePriority* shall be set within the range [0, 255].

Syntax description:

```
SetCachePriority          -->   Target,
                                NewCachePriority
Target                    -->   GenericObjectReference
NewCachePriority          -->   GenericInteger
```

9.5 Formal description

```
Group Class               --> Root Class,
                              StandardIdentifier?,
                              StandardVersion?,
                              ObjectInformation?,
                              OnStartUp?,
                              OnCloseDown?,
                              OriginalGroupCachePriority?,
                              Items?
StandardIdentifier        --> Joint ISO ITU (2), MHEG (19)
StandardVersion           --> INTEGER
ObjectInformation         --> OctetString
```

```
OnStartUp                    --> Action Class
OnCloseDown                  --> Action Class
OriginalGroupCachePriority   --> INTEGER
Items                        --> Item+
Item                         --> ResidentProgram Class |
                                 RemoteProgram Class |
                                 InterchangedProgram Class |
                                 Palette Class |
                                 Font Class |
                                 CursorShape Class |
                                 BooleanVariable Class |
                                 IntegerVariable Class |
                                 OctetStringVariable Class |
                                 ObjectRefVariable Class |
                                 ContentRefVariable Class |
                                 Link Class |
                                 Stream Class |
                                 Bitmap Class |
                                 LineArt Class |
                                 DynamicLineArt Class |
                                 Rectangle Class |
                                 Hotspot Class |
                                 SwitchButton Class |
                                 PushButton Class |
                                 Text Class |
                                 EntryField Class |
                                 HyperText Class |
                                 Slider Class |
                                 TokenGroup Class |
                                 ListGroup Class
```

10 Application Class

Description	Defines a set of Ingredient objects, which are shared within an application scope.
Base class	Group
Subclasses	None
Status	Concrete class

10.1 Attributes

This subclause defines inherited, exchanged and internal attributes for this class.

10.1.1 Inherited attributes

This class has all the attributes of its base class, with identical semantics.

10.1.2 Own exchanged attributes

This class defines the following additional exchanged attributes:

OnSpawnCloseDown — Action object to be executed when the Application is closed by opening another Application via the Spawn action. It may be useful for instance in order to store information that will be used when restarting this Application.

OnSpawnCloseDown may be invoked only by the Spawn action (see Spawn).

- Optional inclusion of an Action object.
- Default value: None.

OnRestart — Action object to be executed when the Application is restarted. This may be useful, for instance, in order to retrieve information that was stored by OnSpawnCloseDown.

OnRestart may be invoked only by the Quit action (see Quit).

- Optional inclusion of an Action object.
- Default value: None.

DefaultAttributes — Defines default attributes Application-wide that shall be used as default values when a corresponding Ingredient attribute is not specified. The following attribute defaults can be set:

CharacterSet — Default character set, or set of character sets, for text rendering throughout the application, except when otherwise specified. This Integer shall be encoded with a value representing the set. The application domain shall define the range of CharacterSet and its semantics.

NOTE - The *CharacterSet* attribute of Application provides the initial character set for all objects containing text in the Application that do not specify it.

- Optional Integer.
- Default value: None.

BackgroundColour — Default colour to be used to render the background of a text object. This attribute is interpreted as a zero-based index in the colour look-up table defined by the *PaletteRef* attribute, or as a direct colour value, depending on the attribute type.

- Optional Integer or OctetString. An Integer will be interpreted as an index in a Palette; an OctetString will be interpreted as a direct colour value.
- Default value: «transparent».

TextColour — Default colour to be used to render the foreground of a text object. This attribute is interpreted as a zero-based index in the colour look-up table defined by the *PaletteRef* attribute, or as a direct colour value, depending on the attribute type.

- Optional Integer or OctetString. An Integer will be interpreted as an index in a Palette; an OctetString will be interpreted as a direct colour value.
- Default value: Any colour.

Font — Default font to use when presenting a Text object.

The *Font* attribute represents either a name for a font (which is resident in the MHEG-5 engine) or a reference to a Font object.

When no font reference is encoded in Application, the Text object is presented using a default font of the MHEG-5 engine.

- Optional attribute.
- OctetString representing a FontName, or reference to a Font object.
- Default value: Default font.

FontAttributes — Default Font attributes such as style, character size, text colour and background colour.

The exact encoding format of the *FontAttributes* attribute is related to the value of the type of Font object mentioned by the *Font* attribute.

- Optional OctetString.
- Default value: No specific attribute set.

BitmapContentHook — Default hook value for all Bitmap objects.

- Optional Integer.
- Default value: None.

StreamContentHook — Default hook value for all Stream objects.

- Optional Integer.
- Default value: None.

TextContentHook — Default hook value for all Text objects.

- Optional Integer.
- Default value: None.

LineArtContentHook — Default hook value for all LineArt objects.

- Optional Integer.
- Default value: None.

Interchanged-ProgramContentHook
Default hook value for all InterchangedProgram objects.
- Optional Integer.
- Default value: None.

ButtonRefColour
Default ButtonColour colour for Button rendering.
- Optional Integer or OctetString.
- Default value: None.

HighlightRefColour
Default HighlightRefColour for Interactibles.
- Optional Integer or OctetString.
- Default value: None.

SliderRefColour
Default SliderColour colour Slider rendering.
- Optional Integer or OctetString.
- Default value: None.

10.1.3 Own internal attributes

This class defines the following additional internal attributes:

LockCount
Specify whether or not the display screen is in a frozen state.

When this attribute is positive and different from zero, the display shall not reflect any changes made to Visible objects that would normally result in a change in their rendering. However, all those changes shall be reflected at once as soon as the screen is unfrozen. The latter is signalled by the *LockCount* attribute being set to 0.

Audio objects (of a Stream multiplex) will continue to play through a LockScreen action, i.e., they will continue to be heard.

Visible objects that are part of a Stream multiplex shall continue to be played, but any other changes to these objects (e.g., position, volume, etc.) shall not be reflected until the screen is unfrozen; by continue to play is meant that they shall continue to move their *CounterPosition* attribute when the screen is locked, but, the updating of the graphical presentation of these objects during locked screen is optional. On some engines their physical rendering continues running, on some others the image is stopped until the screen is unlocked.

When the screen unlocks, the rendering of the Visible objects shall be compliant with the value of the *CounterPosition* attribute.

- Integer greater than or equal to zero.
- Initial value: 0.

DisplayStack
Ordered list of references to Visible objects indicating how Visibles of the application are organised in graphics layers.

Visibles at the bottom of the *DisplayStack* are displayed in the background of the screen and Visibles at the top of the *DisplayStack* are displayed in the foreground of the screen.

Note that the *DisplayStack* may contain references to inactive Visible objects. In this case, inactive Visible objects simply do not appear on the screen but they remain as valid elements of the *DisplayStack*.

- Ordered list of ObjectReferences to Visible objects.
- Initial value: Empty list.

10.2 Events

This class has the same events as its base class, with identical semantics. In addition, the following events are defined:

EngineEvent This event is generated when a particular event has occurred in the environment of the MHEG-5 engine. This part of ISO/IEC 13522 does not specify any of these events; the application domain may specify the semantics of each of these events and the value of their associated data.

- Associated data: EventTag - Integer.

10.3 Internal behaviours

The following internal behaviours semantics have changed from this object's base class:

Deactivation Execute the following sequence of actions:

1. Apply the CloseConnection action to all opened auxiliary connections.
2. Apply the *Deactivation* behaviour as inherited from the base class.

10.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base class, with identical semantics. In addition, the following applicable MHEG-5 actions are defined:

StorePersistent (*StoreSucceeded*, *InVariables*, *OutFileName*)

Requests the MHEG-5 engine to save data in such a way that it may later be retrieved by the ReadPersistent action.

Data to be saved is passed through a set of Variables referenced by the *InVariables* parameter. These variables may contain Booleans, Integers, OctetStrings, ObjectReferences and ContentReferences.

The data is saved in a file data structure. The *OutFileName* is another parameter of the StorePersistent action. This part of ISO/IEC 13522 does not define the nature, structure, ownership, protection or restriction of the file space. However, StorePersistent and ReadPersistent actions shall use the same file name space.

The effect of the StorePersistent action is synchronous. On the successful completion of the StorePersistent action, the Variable referenced by *StoreSucceeded* shall be set to True, otherwise to False.

Example:
Consider the following elementary action included as part of the OnSpawnCloseDown attribute of the an Application,

:StorePersistent (("myApp" 0) ("myApp" 1) (("scene1" 1) ("scene1" 2)) "myfile.txt")
("myApp" 0) is the ObjectIdentifier of the current Application. ("scene1" 1) and ("scene1" 2) are ObjectIdentifiers of variables that hold information relative to the current Application, e.g. user information. These data will be stored in file named "myfile.txt". The variable ("myApp" 1) refers to a Boolean indicating whether the elementary action succeeded or not.

Data may be recovered at the beginning or restart of this Application (or of another one) to avoid asking the user for the same information twice.

Provisions of use:

- The *Target* object shall be the active Application object.
- *StoreSucceeded* shall refer to an active BooleanVariable object.
- *InVariables* shall be set to a non empty list of references to active Variable objects of any type.

Syntax description:

```
StorePersistent          -->   Target,
                               StoreSucceeded,
                               InVariables,
                               OutFileName
Target                   -->   GenericObjectReference
StoreSucceeded           -->   ObjectReference
InVariables              -->   ObjectReference+
OutFileName              -->   GenericOctetString
```

ReadPersistent (*ReadSucceeded*, *OutVariables*, *InFileName*)

Request the MHEG-5 engine to read data that has been saved by the StorePersistent action.

Data to be read is recovered through a set of Variable objects, referenced by the parameter *OutVariables*.

The data has been stored in a file data structure. The *InFileName* is another parameter of the ReadPersistent action. This part of ISO/IEC 13522 does not define nature, structure, ownership, protection or restriction of the file space. However, StorePersistent and ReadPersistent actions shall use the same file name space.

The effect of the ReadPersistent action is synchronous. At successful completion of the ReadPersistent action, the Variable referenced by *ReadSucceeded* shall be set to True, otherwise to False.

Example:

Consider the following elementary action as part of the *OnRestart* attribute of the an Application,

:ReadPersistent (("myApp" 0) ("myApp" 1) (("scene1" 1) ("scene1" 2)) "myfile.txt")

The content of the variables indicated by ("scene1" 1) and ("scene1" 2) will be set to the values read from the file "myfile.txt" when the Application is launched.

Provisions of use:

- The *Target* object shall be the active Application object.
- *ReadSucceeded* shall refer to an active BooleanVariable object.
- *OutVariables* shall be set to a non empty list of references to active Variable objects.

Syntax description:

```
ReadPersistent          -->   Target,
```

```
                                ReadSucceeded,
                                OutVariables,
                                InFileName
Target                  -->     GenericObjectReference
ReadSucceeded,          -->     ObjectReference
OutVariables            -->     ObjectReference+
InFileName              -->     GenericOctetString
```

Launch

Activate a new application by flushing the currently active one, if any.

Execute synchronously the following sequence of actions:

1. Apply the *Destruction* behaviour of the currently active Scene object, if any.
2. Apply the *Destruction* behaviour of the currently active Application object, if any.
3. Apply the *Activation* behaviour of the Application object to which the Launch action was targeted.

NOTE - Any events that are generated during the execution of these steps are queued and dealt with only after the entire sequence has ended.

Provisions of use:

- The *Target* object shall be a non-available Application object.

Syntax description:

```
Launch                  -->     Target
Target                  -->     GenericObjectReference
```

Spawn

Activate a new application in such a way that the current application is restarted when the new application quits.

Execute synchronously the following sequence of actions:

1. Execute the *OnSpawnCloseDown* Action of the currently active Application object.
2. Store the *GroupIdentifier* of the currently active Application on the application identifier stack, if any.
3. Execute the effect of the Launch action as described above.

The application identifier stack is an optional feature of an MHEG-5 engine. If it is not implemented, or if the application identifier stack is full, this action shall be implemented as the Launch action.

Provisions of use:

- The *Target* object shall be an Application object that is not currently active while there is currently an active Application object.

Syntax description:

```
Spawn                   -->     Target
Target                  -->     GenericObjectReference
```

Quit

Close an application and restart the previous application.

Execute synchronously the following sequence of actions:

1. Apply the *Destruction* behaviour of the currently active Scene object, if any.
2. Apply the *Destruction* behaviour of the target Application object.
3. If the MHEG-5 engine has not implemented an application identifier stack, or if the application identifier stack is empty, the MHEG-5 engine shall then return to an idle state. In all other cases, the following steps shall be performed:
4. Apply the *Activation* behaviour to the Application object whose Group Identifier is on the top of the application identifier stack. Note that this includes executing the *OnStartUp* Action of that object.
5. Remove the top entry from the application identifier stack.
6. Execute the *OnRestart* Action of the newly activated Application object.

Provisions of use:

- The *Target* object shall be the currently active Application object.

Syntax description:

```
Quit                    -->     Target
Target                  -->     GenericObjectReference
```

LockScreen

Freeze the display screen and prevent from reflecting changes to Visible objects.

Execute synchronously the following sequence of actions:

1. Increment the internal attribute *LockCount* by 1.
2. If the *LockCount* attribute is now a strictly positive value, lock the display screen.

Provisions of use:

- The *Target* object shall be the active Application object.

Syntax description:

```
LockScreen              -->     Target
Target                  -->     GenericObjectReference
```

UnlockScreen

This action may refresh the display screen and reflect at once all changes to Visible objects.

Execute synchronously the following sequence of actions:

1. Decrement the internal attribute *LockCount* by 1. If the result is less than zero, set the *LockCount* attribute to 0.
2. If the *LockCount* attribute is equal to 0, refresh the display screen.

Provisions of use:

- The *Target* object shall be the active Application object.

Syntax description:

```
UnlockScreen        -->     Target
Target              -->     GenericObjectReference
```

OpenConnection (*OpenSucceeded*, *Protocol*, *Address*, *ConnectionTag*)

Attempt to open a connection with an entity outside the MHEG-5 engine.

The OpenConnection action has the following parameters:

OpenSucceeded If the OpenConnection action terminates successfully, the Variable referenced by *OpenSucceeded* shall be set to True, otherwise to False.

Protocol Identifier of the protocol to be used when establishing the connection.

Address Address of the counterpart with whom the connection should be made. The coding of this parameter depends on the value of the *Protocol*.

ConnectionTag Integer used to reference the connection within the application.

Provisions of use:

- The *Target* object shall be the active Application object.
- *OpenSucceeded* shall refer to an active BooleanVariable object.

Syntax description:

```
OpenConnection      -->     Target,
                            OpenSucceeded,
                            Protocol,
                            Address,
                            ConnectionTag
Target              -->     GenericObjectReference
OpenSucceeded       -->     ObjectReference
Protocol            -->     GenericOctetString
Address             -->     GenericOctetString
ConnectionTag       -->     GenericInteger
```

CloseConnection (*ConnectionTag*)

Attempt to close a connection with an entity outside the MHEG-5 engine.

The CloseConnection action has the following parameters:

ConnectionTag Integer referencing a connection created by the OpenConnection action.

Provisions of use:

- The *Target* object shall be the active Application object.
- If the connection referenced by *ConnectionTag* is not properly established, the CloseConnection action is ignored.

Syntax description:

CloseConnection	-->	Target, ConnectionTag
Target	-->	GenericObjectReference
ConnectionTag	-->	GenericInteger

GetEngineSupport (*Feature, Answer*)

Return an Boolean that indicates if the MHEG-5 engine implements the specific option or set of options of this part of ISO/IEC 13522. The result of this action is returned in a BooleanVariable referenced by the *Answer* parameter and may be used to adapt the behaviour of the application to the engine's capacities.

Feature is a string encoded by ISO/IEC 646 and describing the option or set of options. Strings allowed are defined below; additional strings may be defined by the application domain. These strings are case sensitive, and integers are to be substituted to the symbols N, W, H, X or Y in parenthesis.

The answer to each of these strings shall be True or False.

AncillaryConnections

(asks whether engine supports ancillary point to point connections. These connections deal with the actions OpenConnection and CloseConnection, and the attribute ConnectionTag in various elementary actions)

ApplicationStacking

(asks whether engine provides support for the Spawn action of the Application class)

Cloning

(asks whether engine supports the Clone action)

FreeMovingCursor

(asks whether engine provides support for the class CursorShape, for the events CursorEnter and CursorLeave, and for the actions GetCursorPosition, SetCursorPosition, and SetCursorShape)

MultipleAudioStreams(N)

(asks whether engine supports at least N simultaneous Audio streams)

MultipleRTGraphicsStreams(N)

(asks whether engine supports at least N simultaneous RTGraphics streams)

MultipleVideoStreams(N)

(asks whether engine supports at least N simultaneous Video streams)

OverlappingVisibles(N)

(asks whether engine supports at least N overlapping Visibles)

Scaling

(asks whether engine supports ScaleBitmap and ScaleVideo actions)

SceneAspectRatio(W,H)

(asks whether engine supports a given aspect ratio. W & H are two integers, W / H is the width/height aspect ratio.)

SceneCoordinateSystem(X,Y)

(asks whether engine supports a given coordinate system. X & Y are two integers defining the coordinate system.)

TrickModes

(asks whether engine supports trick modes for Streams)

Provisions of use:

- The GetEngineSupport action shall be targeted only to the active Application object.
- *Answer* shall refer to an active BooleanVariable object.

Syntax description:

```
GetEngineSupport        -->     Target,
                        -->     Feature,
                                Answer
Target                  -->     GenericObjectReference
Feature                 -->     GenericOctetString
Answer                  -->     ObjectReference
```

10.5 Formal description

```
Application Class       -->  Group Class,
                             OnSpawnCloseDown?,
                             OnRestart?
                             DefaultAttributes?
OnSpawnCloseDown        -->  Action Class
OnRestart               -->  Action Class
DefaultAttributes       -->  DefaultAttribute+
DefaultAttribute        -->  CharacterSet | BackgroundColour |
                             TextColour | Font | FontAttributes
                             | BitmapContentHook |
                             InterchangedProgramContentHook |
                             StreamContentHook |
                             TextContentHook |
                             LineArtContentHook |
                             ButtonRefColour |
                             HighlightRefColour |
                             SliderRefColour
CharacterSet            -->  INTEGER
BackgroundColour        -->  Colour
TextColour              -->  Colour
Font                    -->  OctetString | ObjectReference
FontAttributes          -->  OctetString
BitmapContentHook       -->  INTEGER
StreamContentHook       -->  INTEGER
TextContentHook         -->  INTEGER
LineArtContentHook      -->  INTEGER
InterchangedProgram-    -->  INTEGER
ContentHook
ButtonRefColour         -->  Colour
HighlightRefColour      -->  Colour
SliderRefColour         -->  Colour
```

11 Scene Class

Description	Defines a set of Ingredient objects to be activated together.
Base class	Group
Subclasses	None
Status	Concrete class

11.1 Attributes

This subclause defines inherited, exchanged and internal attributes for this class.

11.1.1 Inherited attributes

This class has all the attributes of its base class, with identical semantics.

11.1.2 Own exchanged attributes

This class defines the following additional exchanged attributes:

InputEventRegister

Register of permissible UserInput events for this Scene.

While this Scene is active, the MHEG-5 engine shall generate only UserInput events that have an associated data consistent with the content of this InputEventRegister.

The contents of InputEventRegisters is not defined by this part of ISO/IEC 13522.

- Integer identifying an InputEventRegister.

Example:
One could define two InputEventRegisters: one dedicated to moving pointer input events (MouseClick, etc.) and another one dedicated to remote control input events (Up, Down, Left, Right, Enter, Quit, etc.). Knowing which type of input events are expected by the scene will allow the MHEG-5 engine to use the physical user devices to generate such events.

SceneCoordinate-System

Size of the coordinate system of this Scene.

This attribute is expressed in numbers of rows and columns.

- Two Integers, expressing x-scene and y-scene sizes.

AspectRatio

Original aspect ratio of the Scene. This attribute is expressed by a width / height ratio.

- Optional rational number.
- Default value: 4 / 3.

MovingCursor

Indicates whether the Scene is expecting a free-moving cursor.

Support by the MHEG-5 engine for free-moving cursors is optional. However, an application domain of MHEG-5 may declare such support to be mandatory. An engine that does not support free-moving cursors shall disregard this attribute. An engine that does support free-moving cursors shall act in the following way:

- When this attribute is False, the engine shall not display a cursor on the screen.
- When this attribute is True, the engine shall display a cursor on the screen. The

user shall be able to move this cursor to all positions within the Scene coordinate space. When the cursor enters (or leaves) the bounding box of an Interactible object, a CursorEnter (CursorLeave) event shall be generated for that Interactible. The engine shall support the SetCursorPosition and GetCursorPosition actions.

The following applies to this attribute in general:

- Optional Boolean.
- Default value: False.

NextScenes — An optional list of OctetStrings, which shall be interpreted as GroupIdentifiers of Scene objects that might be presented after this one, along with a weight factor measuring the likelihood that these scenes are actually presented. The weight factor shall be an Integer in the range [0, 255] with 255 indicating the highest likelihood. It may be used by the MHEG-5 engine to resolve caching or pre-loading conflicts.

11.1.3 Own internal attributes

This class defines the following additional internal attributes:

Timers — List of timers representing the temporal positions where the Scene shall receive *TimerFired* events.

Each timer has a unique identity number within the *Timers* list and a temporal position expressed in milliseconds. The temporal position is measured from the time origin of the timer. A timer shall be created by executing the SetTimer action on the active Scene; The time origin of the timer is by default the position in time where the SetTimer action, that created this timer, is executed; however, if the AbsoluteTime Boolean is encoded and set to True, the time origin of the timer is the position in time where the IsRunning event of the scene is generated.

- Sequence of the following data structures:
  - Timer identifier: Integer
  - Timer position: Integer
  - AbsoluteTime: Optional Boolean, default: False
- Initial value: Empty sequence.

11.2 Events

This class has the same events as its base class, with identical semantics. In addition, the following events are defined:

UserInput — This event shall be generated by the MHEG-5 engine to indicate that user input has occurred.

- Associated data: UserInputEventTag - Integer. The value of the Associated Data shall be consistent with the content of the InputEventRegister attribute.

TimerFired — This event is generated when a timer has fired.

- Associated data: TimerIdentifier - Integer.

11.3 Internal behaviours

This class has the same internal behaviours as its base class, with identical semantics.

11.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base class, with identical semantics. In addition, the following applicable MHEG-5 actions are defined:

TransitionTo (*ConnectionTag*, *TransitionEffect*)

Check that target scene is different from active scene; if it is not, ignore action; if it is, remove the active Scene from the screen and replace it with the target Scene as follows.

Execute synchronously the following sequence of actions:

1. Apply the *Deactivation* behaviour to all active Ingredient objects of the currently active Application object that have the Shared parameter set to False in the reverse order that they are listed in the Items attribute of the application object.
2. Apply the *Deactivation* and *Destruction* behaviours to the Scene object currently active, if any. (This starts the transition effect.)
3. Apply the *Preparation* behaviour to the Scene object to which the TransitionTo action was targeted.
4. Apply the *Activation* behaviour to the Scene object to which the TransitionTo action was targeted. (This stops the transition effect.) The timeline for the new scene starts after generation of the *IsRunning* event.

This action has an optional parameter called *ConnectionTag*. If this parameter is not encoded, the Scene reference shall be resolved within the name space of the active Application object.

If the *ConnectionTag* parameter is encoded, the reference to the target Scene and all ContentReferences made from the target Scene shall be resolved within a name space that is used for communication over a communication link with the tag *ConnectionTag* (see the action OpenConnection in the Application class).

In addition, this action has a *TransitionEffect* parameter, which determines what type of visual transition effect to be implemented when performing the TransitionTo action. Implementing any transition effect is optional for the MHEG-5 engine. The encoding of the TransitionEffect attribute has to be specified by the application domain.

NOTE - for some transition effects the engine may be required to handle the destruction behaviour of Visibles differently from other deactivation (not introduced by TransitionTo). For example, the visual representation of the current scene may be saved in the graphical subsystem for a wipe or push effect, where the first scene is smoothly replaced by the second.

Provisions of use:

- The *Target* object shall be a non-available Scene object.

Syntax description:

```
TransitionTo        -->   Target,
                          ConnectionTag?,
                          TransitionEffect?
Target              -->   GenericObjectReference
ConnectionTag       -->   GenericInteger
TransitionEffect    -->   GenericInteger
```

SetTimer (*TimerId*, *TimerValue*, *AbsoluteTime*)

Update the list of timers of the Scene.

Execute the following sequence of actions:

1. Update the *Timers* internal attribute of the Scene, according to the following rules:
   a) If *TimerId* is the identifier of an existing Timer in the Scene, the new *TimerValue*, replaces the previous one. The parameter *AbsoluteTime* is ignored, in other words an absolute Timer cannot be replaced by a Timer relative to the Scene.
   b) If there is no Timer with identifier *TimerId* in the Scene, insert a new Timer with identifier *TimerId* and values *TimerValue* and *AbsoluteTime* in the Scene. If *AbsoluteTime* is not encoded, it is set to False by default.
   c) If *TimerValue* is not encoded and there is a Timer with identifier *TimerId* in *the Scene*, remove this Timer from the *Timers* list.
   d) If *TimerValue* is not encoded and there is no Timer with identifier *TimerId* in the *Timers* list, discard this action.
2. The active Scene shall receive *TimerFired* events according to the new value of the *Timers* list.

If *AbsoluteTime* is set to True, the *TimerValue* parameter of this action shall be interpreted as a time offset from the time when the Scene is active. Otherwise, the *TimerValue* parameter of this action shall be interpreted as a time offset from the time when the action is invoked. In both cases, it is measured in milliseconds.

If the *TimerValue* parameter is zero and AbsoluteTime is False, the Timer shall be fired immediately.

Removing or changing a Timer does not suppress pending events from the former Timer.

Provisions of use:

- The *Target* object shall be the active Scene object.

Syntax description:

```
SetTimer            -->   Target,
                          TimerId,
                          TimerValue?,
                          AbsoluteTime?
Target              -->   GenericObjectReference
TimerId             -->   GenericInteger
TimerValue          -->   GenericInteger
AbsoluteTime        -->   GenericBoolean
```

SendEvent (*EmulatedEvent-Source*, *EmulatedEvent-Type*, *EmulatedEventData*)

Force the occurrence of an event.

Execute the following sequence of actions:

1. Generate an event corresponding to *EmulatedEventType*, *EmulatedEventSource* and *EmulatedEventData* as if it has been generated in the normal way.
2. Store this event in the synchronous or asynchronous event queue, according to *EmulatedEventType*.

Provisions of use:

- The *Target* object shall be the active Scene object.
- *EmulatedEventSource* shall refer to an MHEG-5 object compatible with *EmulatedEventType*.
- *EmulatedEventData* shall be either a direct value, or a reference to an active Variable object of a type compatible with the associated data of *EmulatedEventType*.

Syntax description:

```
SendEvent               -->     Target,
                                EmulatedEventSource,
                                EmulatedEventType,
                                EmulatedEventData?
Target                  -->     GenericObjectReference
EmulatedEventSource     -->     GenericObjectReference
EmulatedEventType       -->     IsAvailable | ContentAvailable |
                                IsDeleted | IsRunning | IsStopped
                                | UserInput | AnchorFired |
                                TimerFired | AsynchStopped |
                                InteractionCompleted | TestEvent
                                | TokenMovedFrom | TokenMovedTo |
                                FirstItemPresented |
                                LastItemPresented | HeadItems |
                                TailItems | ItemSelected |
                                ItemDeselected | StreamEvent |
                                StreamPlaying | StreamStopped |
                                CounterTrigger | HighlightOn |
                                HighlightOff | CursorEnter |
                                CursorLeave | IsSelected |
                                IsDeselected | EntryFieldFull
EmulatedEventData       -->     GenericBoolean | GenericInteger |
                                GenericOctetString
```

SetCursorShape (*NewCursorShape*)

Set the shape of the free moving cursor.

This action shall have an effect only if the free-moving cursor option is implemented by the MHEG-5 engine.

If the *NewCursorShape* parameter is not encoded, the cursor is removed from the Scene.

Provisions of use:

- The *Target* object shall be the active Scene object.
- *NewCursorShape* shall refer to an active CursorShape object.

Syntax description:

```
SetCursorShape          -->     Target,
                                NewCursorShape?
Target                  -->     GenericObjectReference
NewCursorShape          -->     GenericObjectReference
```

SetCursorPosition (*XCursor*, *YCursor*)

Set the position of the free moving cursor.

This action shall have an effect only if the free-moving cursor option is implemented by the MHEG-5 engine.

Execute the following sequence of actions:

1. Set the position of the cursor pointer within the coordinate space of the Scene.
2. Generate *CursorLeave* and *CursorEnter* events if Interactible objects are affected by the effect of this action.

If an Interactible B overlaps another one A, a SetCursorPosition from a point in A (not in B) to a point within the overlapping area shall generate a *CursorLeave(A)* and a *CursorEnter(B).*

Provisions of use:

- The *Target* object shall be the active Scene object.
- *XCursor* and *YCursor* shall correspond to a location within the rectangle defined by the SceneCoordinateSystem attribute of the active Scene.

Syntax description:

```
SetCursorPosition     -->    Target,
                             XCursor,
                             YCursor
Target                -->    GenericObjectReference
XCursor               -->    GenericInteger
YCursor               -->    GenericInteger
```

GetCursorPosition (*XOut*, *YOut*)

Set the Variables referenced by *XOut* and *YOut* to the location of the free moving cursor within the coordinate space of the Scene.

This action shall have an effect only if the free-moving cursor option is implemented by the MHEG-5 engine.

Provisions of use:

- The *Target* object shall be the active Scene object.
- *XOut* and *YOut* shall refer to active IntegerVariable objects.

Syntax description:

```
GetCursorPosition     -->    Target,
                             XOut,
                             YOut
Target                -->    GenericObjectReference
XOut                  -->    ObjectReference
YOut                  -->    ObjectReference
```

11.5 Formal description

```
Scene Class                --> Group Class,
                               InputEventRegister,
                               SceneCoordinateSystem,
                               AspectRatio?,
                               MovingCursor?,
                               NextScenes?
InputEventRegister         --> INTEGER
SceneCoordinateSystem      --> XScene, YScene
XScene,                    --> INTEGER
YScene                     --> INTEGER
AspectRatio                --> Width, Height
Width                      --> INTEGER
Height                     --> INTEGER
MovingCursor               --> BOOLEAN
NextScenes                 --> NextScene+
NextScene                  --> SceneRef,
                               SceneWeight
SceneRef                   --> OctetString
SceneWeight                --> INTEGER
```

12 Ingredient Class

Description	Defines the functionality associated with classes that make up Scenes and Applications.
Base class	Root
Subclasses	Link, Program, Palette, Font, CursorShape, Variable, Presentable
Status	Abstract class

12.1 Attributes

This subclause defines inherited, exchanged and internal attributes for this class.

12.1.1 Inherited attributes

This class has all the attributes of its base class, with the following constraints:

Attribute Name	Defined in	Constraints and Requirements
ObjectIdentifier	Root	The ObjectNumber part of this attribute shall be unique within the group this object belongs to and shall not be 0. If the GroupIdentifier is encoded, it shall be set to the GroupIdentifier of the group this object belongs to.

12.1.2 Own exchanged attributes

This class defines the following additional exchanged attributes:

InitiallyActive — This parameter is used to determine which objects in a Scene or an Application are initially active.

- Optional Boolean.
- Default value: True.

ContentHook — Determine the encoding format of the data included or referenced by the *Content* attribute.

- Optional Integer.
- Default value: Depending on subclasses, the value encoded by Application in one of the following attributes: BitmapContentHook, LineArtContentHook, InterchangedProgramContentHook, StreamContentHook and TextContentHook.

OriginalContent — Value of the *Content* internal attribute at preparation.

This attribute contains either included data or a reference to an external data source.

Included data is encoded directly within an OctetString.

A reference to an external data source is composed of:

1. A ContentReference, which is an OctetString referencing an external piece of data.
   - Data type: OctetString.
2. An optional ContentSize, which represents the size in bytes of the external

source. This attribute might be used by the MHEG-5 engine to determine how many resources are requested to render the Ingredient. It is the responsibility of the application to ensure the compatibility of the ContentSize attribute with the actual size of the external source of data.

- ContentSize is an optional attribute.
- Data type: Integer.
- Default value: None.

Example: Consider a Bitmap object whose OriginalContent attribute is set to a reference to an external data file. Its ContentSize attribute should contain an estimation of the size of the external data.

3. An optional ContentCachePriority, which represents the relevance of caching this external data source. This attribute might be compared by the MHEG-5 engine to other ContentCachePriority attributes to determine which external data source has the highest priority of being required again by the application. A higher value indicates a higher level of priority. It is the responsibility of the application to keep these priorities to a consistent range. The MHEG-5 engine is recommended to cache external data sources with higher priority in preference to data sources with lower priority.

   - ContentCachePriority is an optional attribute.
   - Data type: Integer within the range [0, 255].
   - Default value: 127.
   - Specific value: 0 means caching is not allowed for external content data referenced by this Ingredient.

- OriginalContent is an optional attribute.
- Data type: ReferencedContent or IncludedContent.
- Default value: None.

Shared — Indicate whether the Ingredient object is intended for continuous presentation across a Scene transition. This is used to prevent destroying objects that are used in consecutive scenes. Specifically, when a TransitionTo action is targeted to a Scene *B* while Scene *A* is active, all active ingredients of the active Application object are automatically deactivated except those that have the Shared attribute set to True.

Provisions of use:

- If the Ingredient object is an item of a Scene, Shared shall not be encoded.

Synopsis:

- Optional Boolean.
- Default value: False.

12.1.3 Own internal attributes

This class defines the following additional internal attributes:

Content — This attribute contains either included data or a reference to an external data source.

Included data is encoded directly within an OctetString.

A reference to an external data source is composed of:

1. A ContentReference, which is an OctetString referencing an external piece of data.
   - Data type: OctetString.
   - Initial value: ContentSize of OriginalContent attribute.
2. An optional ContentSize, which represents the size in bytes of the external source.
   - ContentSize is an optional attribute.
   - Data type: Integer.
   - Initial value: ContentSize of OriginalContent attribute.
3. An optional ContentCachePriority, which represents the relevance of caching this external data source.
   - ContentCachePriority is an optional attribute.
   - Data type: Integer within the range [0, 255].
   - Initial value: ContentCachePriority of OriginalContent attribute.
   - Specific value: 0 means caching is not allowed for external content data referenced by this Ingredient.

The *Content* attribute shall not be defined for classes specifying that OriginalContent shall not be encoded,

- Optional attribute.
- Data type: ReferencedContent or IncludedContent.
- Initial value: Value of OriginalContent.

12.2 Events

This class has the same events as its base class, with identical semantics.

12.3 Internal behaviours

The following internal behaviours semantics have changed from this object's base class:

Destruction Execute the following sequence of actions:

1. If the *Content* attribute is set to a reference to external data source and if ContentCachePriority is set to 0, the MHEG-5 engine shall free all resources allocated to the external content data of the Ingredient.
2. If the *Content* attribute is set to a reference to external data source and if ContentCachePriority is different from 0, the MHEG-5 engine may free all resources allocated to the external content data of the Ingredient, or cache it.
3. Apply the *Destruction* behaviour as inherited from the Root class.

12.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base class, with identical semantics. In addition, the following applicable MHEG-5 actions are defined:

SetData (*NewContent*)

Set the *Content* attribute of the target Ingredient to *NewContent*.

Provisions of use:

- The *Target* object shall be an available Ingredient object.
- The ContentHook of the target Ingredient object shall be encoded.
- Data included or referenced by *NewContent* shall have the encoding format determined by ContentHook of the target Ingredient.
- If *Content* is currently set to included data, *NewContent* shall be set or refer to included data.
- If *Content* is currently set to a reference to an external data source, then *NewContent* shall be set or refer to a reference to an external data source.

Syntax description:

```
SetData                    -->  Target,
                                NewContent

Target                     -->  GenericObjectReference
NewContent                 -->  NewIncludedContent |
                                NewReferencedContent
NewIncludedContent         -->  GenericOctetString
NewReferencedContent       -->  NewContentReference,
                                NewContentSize?,
                                NewContentCachePriority?
NewContentReference        -->  GenericContentReference
NewContentSize             -->  GenericInteger
NewContentCachePriority    -->  GenericInteger
```

Clone (*CloneRefVar*)

If the engine supports the Cloning option the effect of this action is described below. Engines that do not support the Cloning option shall ignore this action.

This action copies the *Target* adding the copy to the same group that contains the *Target* using a unique ObjectReference obtained from the engine. The ObjectReference referring to the copy is returned in the ObjectRefVariable referenced by *CloneRefVar*.

Execute the following sequence of actions:

1. Determine a unique ObjectReference within the same group as the *Target*.
2. Create a copy of the *Target*, taking into account only the exchanged attributes and not the internal attributes. The *ObjectIdentifier* attribute inherited from the Root class is not copied, but set to the ObjectReference determined in step 1.
3. Add the copy to the group containing the Target object.
4. Set the ObjectRefVariable referenced by *CloneRefVar* to the ObjectReference determined in step 1.
5. Apply the *Preparation* behaviour to the copy.

An object created using this action is deactivated/destroyed when the group that contains that object is deactivated/destroyed. The objects are deactivated/destroyed

along with other static Ingredients in the reverse order of creation. A dynamically created Ingredient can also be destroyed using the Unload elementary action if its *Content* attribute is not Null.

Provision of use:

- The *Target* object shall be an available Ingredient.
- *CloneRefVar* shall be an active ObjectRefVariable.

Syntax description:

```
Clone                   -->     Target,
                                CloneRefVar
Target                  -->     GenericObjectReference
CloneRefVar             -->     ObjectReference
```

Preload

Prepares an Ingredient and provides a hint to the MHEG-5 engine to prepare the content data of an Ingredient for future use.

Execute the following sequence of actions:

1. Apply the *Preparation* behaviour.
2. The MHEG-5 engine may optionally retrieve and/or decode the content data associated with the target Ingredient object.

Provisions of use:

- The *Target* object shall be non-available Ingredient object.
- The *Content* attribute of the target Ingredient shall be different from Null.

Syntax description:

```
Preload                 -->     Target
Target                  -->     GenericObjectReference
```

Unload

Destroys an Ingredient and provides a hint to the MHEG-5 engine to free resources allocated to an Ingredient.

Execute the following sequence of actions:

1. Apply the *Destruction* behaviour.

Provisions of use:

- The *Target* object shall be an available and inactive Ingredient object.
- The *Content* attribute of the target Ingredient shall be different from Null.

Syntax description:

```
Unload                  -->     Target
Target                  -->     GenericObjectReference
```

12.5 Formal description

```
Ingredient Class            --> Root Class,
                                InitiallyActive?,
                                ContentHook?,
                                OriginalContent?,
                                Shared?
InitiallyActive             --> BOOLEAN
ContentHook                 --> INTEGER
OriginalContent             --> IncludedContent |
                                ReferencedContent
IncludedContent             --> OctetString
ReferencedContent           --> ContentReference,
                                ContentSize?,
                                ContentCachePriority?
ContentSize                 --> INTEGER
ContentCachePriority        --> INTEGER
Shared                      --> BOOLEAN
```

13 Link Class

Description	Defines the functionality associated with reacting to events by performing a sequence of elementary actions.
Base class	Ingredient
Subclasses	None
Status	Concrete class

13.1 Attributes

This subclause defines inherited, exchanged and internal attributes for this class.

13.1.1 Inherited attributes

This class has all the attributes of its base class, with the following constraints:

Attribute Name	Defined in	Constraints and Requirements
ContentHook	Ingredient	This attribute shall not be encoded for this class.
OriginalContent	Ingredient	This attribute shall not be encoded for this class.

13.1.2 Own exchanged attributes

This class defines the following additional exchanged attributes:

LinkCondition

The *LinkCondition* consists of an *EventSource,* an *EventType* and an *EventData.*

When an event emanates from an object, the MHEG-5 engine shall fire a specific Link if and only if

- the Link is active;
- the EventSource is equal to the object reference of the object from which the event emanated (the GroupIdentifier defaults to the GroupIdentifier of the Group in which the Link itself is included);
- the EventType is equal to the type of event that occurred; and
- *either,* the EventData is equal to the data value provided with the event, *or* the EventData is not encoded.

NOTE - The type of data passed with each event is described in clause 53.

The firing of a Link object leads to the execution of its LinkEffect.

LinkEffect

Inclusion of an Action object.

When the Link fires, the elementary actions within this Action object are executed in synchronous order.

13.1.3 Own internal attributes

This class defines no additional internal attribute.

13.2 Events

This class has the same events as its base class, with identical semantics.

13.3 Internal behaviours

The following internal behaviours semantics have changed from this object's base class:

Activation Execute the following sequence of actions:

1. Apply the *Activation* behaviour as inherited from the base class.
2. Make the Link object receptive to events that fulfil its LinkCondition.
3. Set the *RunningStatus* of the Link to True.
4. Generate an *IsRunning* event.

Deactivation Execute the following sequence of actions:

1. Put the Link object in an inactive state so that it is not receptive to events.
2. Apply the *Deactivation* behaviour as inherited from the base class.

13.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base class, with identical semantics. In addition, the following applicable MHEG-5 actions are defined:

Activate Make a Link receptive to events that meet its LinkCondition.

Execute the following actions:

1. If the target Link object is active, disregard this action.
2. If the target Link is inactive, apply the *Activation* behaviour of the Link object.

Syntax description:

```
Activate        -->    Target
Target          -->    GenericObjectReference
```

Deactivate Make a Link not receptive to events.

Execute the following actions:

1. If the target Link object is inactive, disregard this action.
2. If the target Link is active, apply the *Deactivation* behaviour of the Link object.

Provisions of use:

- The *Target* object shall be an available Link object.

Syntax description:

```
Deactivate      -->    Target
Target          -->    GenericObjectReference
```

13.5 Formal description

```
Link Class              -->  Ingredient Class,
                             LinkCondition,
                             LinkEffect
LinkCondition           -->  EventSource,
                             EventType,
                             EventData?
LinkEffect              -->  Action Class
EventSource             -->  ObjectReference
EventType               -->  IsAvailable | ContentAvailable |
                             IsDeleted | IsRunning | IsStopped
                             | UserInput | AnchorFired |
                             TimerFired | AsynchStopped |
                             InteractionCompleted | TestEvent
                             | TokenMovedFrom | TokenMovedTo |
                             FirstItemPresented |
                             LastItemPresented | HeadItems |
                             TailItems | ItemSelected |
                             ItemDeselected | StreamEvent |
                             StreamPlaying | StreamStopped |
                             CounterTrigger | HighlightOn |
                             HighlightOff | CursorEnter |
                             CursorLeave | IsSelected |
                             IsDeselected / EntryFieldFull |
                             EngineEvent
EventData               -->  OctetString | BOOLEAN | INTEGER
```

14 Program Class

Description	Defines means to handle execution of external pieces of procedural code.
Base class	Ingredient
Subclasses	RemoteProgram, ResidentProgram, InterchangedProgram
Status	Abstract class

14.1 Attributes

This subclause defines inherited, exchanged and internal attributes for this class.

14.1.1 Inherited attributes

This class has all the attributes of its base class, with the following constraints:

Attribute Name	Defined in	Constraints and Requirements
InitiallyActive	Ingredient	This attribute is mandatory for this class and shall be set to False.

14.1.2 Own exchanged attributes

This class defines the following additional exchanged attributes:

Name — Name of the external procedural code to be called when the Program object is activated.

The mapping of Name to the actual name of the external procedural code is not defined by this part of ISO/IEC 13522.

- OctetString.

InitiallyAvailable — This parameter is used to determine which Programs in a Scene or an Application are initially prepared.

- Optional Boolean.
- Default value: True.

14.1.3 Own internal attributes

This class defines no additional internal attribute.

14.2 Events

This class has the same events as its base class, with identical semantics. In addition, the following events are defined:

AsynchStopped — This event shall be generated when a Program object, that is executed via the Fork action, has terminated its execution.

- No associated data.

14.3 Internal behaviours

The following internal behaviours semantics have changed from this object's base class:

Activation

1. If not done during preparation, locate the external procedural code by using the Name attribute.
2. If the external procedural code is not found, disregard this action. Otherwise,
3. Set the parameters of the external procedural code of the Program as indicated by *Parameters*.
4. Apply the *Activation* behaviour as inherited from the base class.
5. Start execution of the external procedural code synchronously or asynchronously according to the action invoking the execution.
6. Set the *RunningStatus* attribute to True.
7. Generate an *IsRunning* event.

Deactivation

If the *RunningStatus* attribute is False, ignore this action. Otherwise, execute the following sequence of actions:

1. Force the end of execution of the Program.
2. Apply *Deactivation* behaviour as inherited from the base class.

NOTE - The *Deactivation* behaviour generates an IsStopped event, as defined in Root.

14.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base class, with identical semantics. In addition, the following applicable MHEG-5 actions are defined:

SetData

This action shall not be applied to an object of any subclasses of the Program class.

Call (*CallSucceeded*, *Parameters*)

Request execution of an external piece of procedural code and wait for the end of execution.

Execute the following sequence of actions:

1. If the Program is non-available, apply the *Preparation* behaviour.
2. If the Program is active, disregard this action. Otherwise,
3. Apply the *Activation* behaviour.
4. Wait for the execution of the external procedural code to finish. If the Program finishes abnormally, set the Variable referenced by *CallSucceeded* to False; otherwise, set it to True.
5. Set the value of the Variables referenced by *Parameters* to the values returned by the Program (these may be invalid if *CallSucceeded* is False).
6. Apply the *Deactivation* behaviour.

Provisions of use:

- *CallSucceeded* shall be set to an active BooleanVariable object.
- *Parameters* shall be set to a list of values corresponding to the expected parameters of the external procedural code. The order of the *Parameters* list shall correspond to the order of parameters of the procedural code. Parameters passed

by value shall be set directly to the corresponding value. Parameters passed by reference shall be passed via active Variable objects of the corresponding data type.

Syntax description:

```
Call              -->       Target,
                            CallSucceeded,
                            Parameters?
Target            -->       GenericObjectReference
CallSucceeded     -->       ObjectReference
Parameters        -->       Parameter+
Parameter         -->       GenericBoolean | GenericInteger |
                            GenericOctetString |
                            GenericObjectReference |
                            GenericContentReference
```

Fork (*ForkSucceeded*, *Parameters*)

Request execution of an external piece of procedural code without waiting for the end of execution.

Execute the following sequence of actions:

1. If the Program is non-available, apply the *Preparation* behaviour.
2. If the Program is active, disregard this action. Otherwise,
3. Apply the *Activation* behaviour.
4. Pass control back to the MHEG-5 engine without waiting for the execution of the external procedural code to finish.

When the execution of the external procedural code finishes, execute the following sequence of actions:

1. If the Program finishes abnormally, set the Variable referenced by *ForkSucceeded* to False; otherwise, set it to True.
2. Set the value of the Variables referenced by *Parameters* to the values returned by the Program (these may be invalid if *ForkSucceeded* is False).
3. Apply the *Deactivation* behaviour.
4. Generate an *AsynchStopped* event.

NOTE - Parameters may be modified by the Program, in that case, these parameters are not defined until the Program is normally finished, that is until an *AsynchStopped* is generated.

Provisions of use:

- *ForkSucceeded* shall be set to an active BooleanVariable object.
- *Parameters* shall be set to a list of values corresponding to the expected parameters of the external procedural code. The order of the *Parameters* list shall correspond to the order of parameters of the procedural code. Parameters passed by value shall be set directly to the corresponding value. Parameters passed by reference shall be passed via active Variable objects of the corresponding data type.

Syntax description:

```
Fork              -->       Target,
```

```
                                ForkSucceeded,
                                Parameters?
Target              -->         GenericObjectReference
ForkSucceeded       -->         ObjectReference
Parameters          -->          Parameter+
Parameter           -->         GenericBoolean | GenericInteger |
                                GenericOctetString |
                                GenericObjectReference |
                                GenericContentReference
```

Stop Interrupt the execution of an external piece of procedural code.

Execute the following sequence of actions:

1. If the Program is inactive, disregard this action. Otherwise,
2. Apply the *Deactivation* behaviour.

Provisions of use:

- The *Target* object shall be an available Program object.

Syntax description:

```
Stop                -->  Target
Target              -->  GenericObjectReference
```

14.5 Formal description

```
Program Class       -->  Ingredient Class,
                         Name,
                         InitiallyAvailable?
Name                -->  OctetString
InitiallyAvailable  -->  BOOLEAN
```

15 ResidentProgram Class

Description	Defines means to handle calls to locally executed external procedural code. A ResidentProgram object provides an interface to a piece of procedural code that is local to the device on which the MHEG-5 engine is running. This part of ISO/IEC 13522 does not specify a naming paradigm for such a procedural interface nor the internal semantics of calling such a Program.
Base class	Program
Subclasses	None
Status	Concrete class

15.1 Attributes

This subclause defines inherited, exchanged and internal attributes for this class.

15.1.1 Inherited attributes

This class has all the attributes of its base class, with the following constraints:

Attribute Name	Defined in	Constraints and Requirements
ContentHook	Ingredient	This attribute shall not be encoded for this class.
OriginalContent	Ingredient	This attribute shall not be encoded for this class.

15.1.2 Own exchanged attributes

This class defines no additional exchanged attributes:

15.1.3 Own internal attributes

This class defines no additional internal attribute.

15.2 Events

This class has the same events as its base class, with identical semantics.

15.3 Internal behaviours

This class has the same internal behaviours as its base class, with identical semantics.

15.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base class, with identical semantics.

15.5 Formal description

```
ResidentProgram Class    -->  Program Class
```

16 RemoteProgram Class

Description	Defines means to handle calls to remotely executed procedural code. A RemoteProgram object provides an interface to a piece of procedural code that is to be run at a location outside the device on which the MHEG-5 engine is running. The location of the remote Program object is provided by means of the ProgramConnectionTag attribute.
Base class	Program
Subclasses	None
Status	Concrete class

16.1 Attributes

This subclause defines inherited, exchanged and internal attributes for this class.

16.1.1 Inherited attributes

This class has all the attributes of its base class, with the following constraints:

Attribute Name	Defined in	Constraints and Requirements
ContentHook	Ingredient	This attribute shall not be encoded for this class.
OriginalContent	Ingredient	This attribute shall not be encoded for this class.

16.1.2 Own exchanged attributes

This class defines the following additional exchanged attributes:

ProgramConnection-Tag

Tag of the connection used to locate the remote procedural code to be called when the Program object is activated. ProgramConnectionTag is an identifier of a connection opened by the OpenConnection action of the Application class.

ProgramConnectionTag is optional. When it is not encoded, the external procedural code is located relatively to the default name space of the application.

- Optional Integer.
- Default value: None.

16.1.3 Own internal attributes

This class defines no additional internal attribute.

16.2 Events

This class has the same events as its base class, with identical semantics.

16.3 Internal behaviours

The following internal behaviours semantics have changed from this object's base class:

Activation

1. If not done during preparation, locate the remote external procedural code by using the Name and ProgramConnectionTag attributes.

2. If the remote external procedural code is not found, disregard this action. Otherwise,
3. Apply the *Activation* behaviour as inherited from the base class.

16.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base class, with identical semantics.

16.5 Formal description

```
RemoteProgram Class      -->  Program Class,
                              ProgramConnectionTag?
ProgramConnectionTag     -->  INTEGER
```

17 InterchangedProgram Class

Description	Defines means to handle program code interchanged as the OriginalContent of a InterchangedProgram object and executed or interpreted on the same device as the MHEG-5 engine. This part of ISO/IEC 13522 does not specify how the procedural code is executed or interpreted on the device on which the MHEG-5 engine is running.
Base class	Program
Subclasses	None
Status	Concrete class

17.1 Attributes

This subclause defines inherited, exchanged and internal attributes for this class.

17.1.1 Inherited attributes

This class has all the attributes of its base class, with the following constraints:

Attribute Name	Defined in	Constraints and Requirements
ContentHook	Ingredient	Either this attribute or its corresponding default attribute in the Application class (InterchangedProgramContentHook) is mandatory for this class.
OriginalContent	Ingredient	This attribute is mandatory for this class.

17.1.2 Own exchanged attributes

This class defines no additional exchanged attribute.

17.1.3 Own internal attributes

This class defines no additional internal attribute.

17.2 Events

This class has the same events as its base class, with identical semantics.

17.3 Internal behaviours

This class has the same internal behaviours as its base class, with identical semantics.

17.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base class, with identical semantics.

17.5 Formal description

```
InterchangedProgram Class     -->  Program Class
```

18 Palette Class

Description	Defines a class to represent a colour look-up table.
Base class	Ingredient
Subclasses	None
Status	Concrete class

18.1 Attributes

This subclause defines inherited, exchanged and internal attributes for this class.

18.1.1 Inherited attributes

This class has all the attributes of its base class, with the following constraints:

Attribute Name	Defined in	Constraints and Requirements
ContentHook	Ingredient	This attribute is mandatory for this class.
OriginalContent	Ingredient	This attribute is mandatory for this class.
InitiallyActive	Ingredient	If encoded, this attribute shall be set to True for this class.

18.1.2 Own exchanged attributes

This class defines no additional exchanged attribute.

18.1.3 Own internal attributes

This class defines no additional internal attribute.

18.2 Events

This class has the same events as its base class, with identical semantics.

18.3 Internal behaviours

This class has the same internal behaviours as its base class, with identical semantics.

18.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base class, with identical semantics.

18.5 Formal description

```
Palette Class              -->  Ingredient Class
```

19 Font Class

Description	Defines a class to represent a character font used for rendering text objects.
Base class	Ingredient
Subclasses	None
Status	Concrete class

19.1 Attributes

This subclause defines inherited, exchanged and internal attributes for this class.

19.1.1 Inherited attributes

This class has all the attributes of its base class, with the following constraints:

Attribute Name	Defined in	Constraints and Requirements
ContentHook	Ingredient	This attribute is mandatory for this class.
OriginalContent	Ingredient	This attribute is mandatory for this class.
InitiallyActive	Ingredient	If encoded, this attribute shall be set to True for this class.

19.1.2 Own exchanged attributes

This class defines no additional exchanged attribute.

19.1.3 Own internal attributes

This class defines no additional internal attribute.

19.2 Events

This class has the same events as its base class, with identical semantics.

19.3 Internal behaviours

This class has the same internal behaviours as its base class, with identical semantics.

19.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base class, with identical semantics.

19.5 Formal description

```
Font Class                    -->  Ingredient Class
```

20 CursorShape Class

Description	Defines encapsulation for the data structures used to represent a free-moving cursor.
Base class	Ingredient
Subclasses	None
Status	Concrete class

20.1 Attributes

This subclause defines inherited, exchanged and internal attributes for this class.

20.1.1 Inherited attributes

This class has all the attributes of its base class, with the following constraints:

Attribute Name	Defined in	Constraints and Requirements
ContentHook	Ingredient	This attribute is mandatory for this class.
OriginalContent	Ingredient	This attribute is mandatory for this class.
InitiallyActive	Ingredient	If encoded, this attribute shall be set to True for this class.

20.1.2 Own exchanged attributes

This class defines no additional exchanged attribute.

20.1.3 Own internal attributes

This class defines no additional internal attribute.

20.2 Events

This class has the same events as its base class, with identical semantics.

20.3 Internal behaviours

This class has the same internal behaviours as its base class, with identical semantics.

20.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base class, with identical semantics.

20.5 Formal description

```
CursorShape Class        -->  Ingredient Class
```

21 Variable Class

Description	Defines a variable within the context of a Group object.
Base class	Ingredient
Subclasses	BooleanVariable, IntegerVariable, OctetStringVariable, ObjectReferenceVariable, ContentRefVariable
Status	Abstract class

21.1 Attributes

This subclause defines inherited, exchanged and internal attributes for this class.

21.1.1 Inherited attributes

This class has all the attributes of its base class, with the following constraints:

Attribute Name	Defined in	Constraints and Requirements
ContentHook	Ingredient	This attribute shall not be encoded for this class.
OriginalContent	Ingredient	This attribute shall not be encoded for this class.
InitiallyActive	Ingredient	If encoded, this attribute shall be set to True for this class.

21.1.2 Own exchanged attributes

This class defines the following additional exchanged attributes:

OriginalValue — Value of the variable when it is first prepared.

The *OriginalValue* attribute shall be of one of these types: Boolean, Integer, OctetString, ObjectReference, or ContentReference.

21.1.3 Own internal attributes

This class defines the following additional internal attributes:

Value — Current value of the variable.

The *Value* attribute may be of one of these types: Boolean, Integer, OctetString, ObjectReference, or ContentReference. The only elementary action that may modify this attribute is the SetVariable action.

- Initial value: Value of the OriginalValue attribute.

21.2 Events

This class has the same events as its base class, with identical semantics. In addition, the following events are defined:

TestEvent — This event shall be generated by the MHEG-5 engine to indicate that a subclass of Variable has been tested.

- Associated data: Boolean. The result of the comparison between the variable and the parameter from the TestVariable action.

21.3 Internal behaviours

The following internal behaviours semantics have changed from this object's base class:

Activation Execute the following sequence of actions:

1. Apply the *Activation* behaviour as defined in the base class.
2. Set the *RunningStatus* attribute to True and generate an *IsRunning* event.

21.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base class, with identical semantics. In addition, the following applicable MHEG-5 actions are defined:

SetVariable (*NewVariableValue*) Set the *Value* attribute of the *Target* object to *NewVariableValue*.

Provisions of use:

- The *Target* object shall be an active object of one of the following classes: BooleanVariable, IntegerVariable, OctetStringVariable, ObjectRefVariable or ContentRefVariable.
- *NewVariableValue* shall be set or refer to a value that is not necessarily of the same type as the current *Value* attribute of the target Variable. When *NewVariableValue* and the Target object are of two different types, *NewVariableValue* is automatically converted into the class type of Target.

NOTE - More details on conversions are given in subclauses relevant to each Variable subclasses.

Syntax description:

```
SetVariable             -->   Target,
                              NewVariableValue
Target                  -->   GenericObjectReference
NewVariableValue        -->   GenericInteger |
                              GenericBoolean |
                              GenericOctetString |
                              GenericObjectReference |
                              GenericContentReference
```

TestVariable (Target, Operator, ComparisonValue) Execute the following sequence of actions:

1. Compare the *Value* of the variable to the ComparisonValue parameter. The Variable value is the first operand, the ComparisonValue parameter is the second operand of the comparison.
2. Generate the corresponding TestEvent.

- Source: Target of TestVariable,
- Associated data: Boolean.

Provisions of use:

- The *Target* object shall be an active object of one of the following classes:

BooleanVariable, IntegerVariable, OctetStringVariable, ObjectRefVariable or ContentRefVariable.

- The ComparisonValue shall be of corresponding type (GenericBoolean, GenericInteger, GenericOctetString, GenericObjectReference and GenericContentReference respectively). No implicit type conversion is allowed.
- When values are Integer, the Operator shall be an integer in the range 1 to 6 with the following meaning:

  1 means equal, 2 not equal,

  3 strictly less than, 4 less than or equal to,

  5 strictly greater than, 6 greater than or equal to.
- When values are Boolean, OctetString, ObjectReference or ContentReference, the Operator shall be an integer in the range 1 to 2 with the following meaning:

  1 means equal, 2 not equal.

Syntax description:

```
TestVariable          -->   Target,
                            Operator,
                            ComparisonValue
Target                -->   GenericObjectReference
ComparisonValue       -->   GenericInteger |
                            GenericBoolean |
                            GenericOctetString |
                            GenericObjectReference |
                            GenericContentReference
Operator              -->   GenericInteger
```

21.5 Formal description

```
Variable Class        -->   Ingredient Class,
                            OriginalValue
OriginalValue         -->   BOOLEAN | INTEGER | OctetString |
                            ObjectReference | ContentReference
```

22 BooleanVariable Class

Description	Defines a variable of type Boolean within the context of a Group object.
Base class	Variable
Subclasses	None
Status	Concrete class

22.1 Attributes

This subclause defines inherited, exchanged and internal attributes for this class.

22.1.1 Inherited attributes

This class has all the attributes of its base class, with no additional constraints:

22.1.2 Own exchanged attributes

OriginalValue	The OriginalValue attribute shall be a Boolean.

22.1.3 Own internal attributes

This class defines the following additional internal attributes:

Value	• The *Value* attribute shall be a Boolean.

22.2 Events

This class has the same events as its base class, with identical semantics.

22.3 Internal behaviours

This class has the same internal behaviours as its base class, with identical semantics.

22.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base class, with identical semantics. In addition, the following applicable MHEG-5 actions are defined:

SetVariable (*NewVariableValue*)	Provisions of use: • The *NewVariableValue* shall be a GenericBoolean.

22.5 Formal description

```
BooleanVariable Class     --> Variable Class
```

23 IntegerVariable Class

Description	Defines a variable of type Integer within the context of a Group object.
Base class	Variable
Subclasses	None
Status	Concrete class

23.1 Attributes

This subclause defines inherited, exchanged and internal attributes for this class.

23.1.1 Inherited attributes

This class has all the attributes of its base class, with no additional constraints:

23.1.2 Own exchanged attributes

OriginalValue	The OriginalValue attribute shall be an Integer.

23.1.3 Own internal attributes

This class defines the following additional internal attributes:

Value	The *Value* attribute shall be an Integer.

23.2 Events

This class has the same events as its base class, with identical semantics.

23.3 Internal behaviours

This class has the same internal behaviours as its base class, with identical semantics.

23.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base class, with identical semantics. In addition, the following applicable MHEG-5 actions are defined:

SetVariable (*NewVariableValue*)	Provisions of use:
	• The *NewVariableValue* shall be of one of the following types: Integer or OctetString.
	  • When the *NewVariableValue* is of type Integer, the value of the Target variable is set to the value of *NewVariableValue.*
	  • When the *NewVariableValue* is of type OctetString, it is first converted to an Integer, then value of the Target variable is set to that Integer value. Rules for conversion:
	  1. OctetString conversion shall make use of the Application CharacterSet attribute. If several character sets are listed, conversion is made on characters as long as they are "numeric" in

their code set.

2. Conversion shall consider that the OctetString represents the Integer in base 10.
3. OctetString conversion shall be made on the first numeric characters (in ISO/IEC 646: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9) and shall stop as soon as a non numeric character is encountered (with exception of the minus sign in first position, as detailed in rule 4 below).
   For instance an OctetString representing "123abc" will be converted into the Integer 123.
4. Minus sign is allowed as first character of an OctetString.
   For instance OctetString representing "-123" will be converted into Integer -123, but "12-345" will be converted into Integer 12 according to rule 3.

Add (*Value*)

Add *Target* Variable to *Value*. Target variable is the first operand of the infix operation. Result is stored in Target Variable.

Provision of use:

- The *Target* object shall be an active IntegerVariable object.

Syntax description:

```
Add                 -->   Target,
                          Value

Target              -->   GenericObjectReference
Value               -->   GenericInteger
```

Subtract (*Value*)

Subtract *Value* from *Target* Variable. Target variable is the first operand of the infix operation. Result is stored in Target Variable.

Provision of use:

- The *Target* object shall be an active IntegerVariable object.

Syntax description:

```
Subtract            -->   Target,
                          Value

Target              -->   GenericObjectReference
Value               -->   GenericInteger
```

Multiply (*Value*)

Multiplies *Target* Variable by *Value*. Target variable is the first operand of the infix operation. Result is stored in Target Variable.

Provision of use:

- The *Target* object shall be an active IntegerVariable object.

Syntax description:

```
Multiply            -->   Target,
```

```
                                Value
Target                  -->     GenericObjectReference
Value                   -->     GenericInteger
```

Divide (*Value*)

Divides *Target* Variable by *Value*. Target variable is the first operand of the infix operation. Result is stored in Target Variable. When the result is not an integer value, rounding is made towards 0.

Provision of use:

- The *Target* object shall be an active IntegerVariable object.

Syntax description:

```
Divide                  -->     Target,
                                Value
Target                  -->     GenericObjectReference
Value                   -->     GenericInteger
```

Modulo (*Value*)

Returns the remainder modulo *Value* of *Target* - as defined by usual integer arithmetic rules, that is to say that for any integers a and b the following equality is satisfied:

(a DIV b) * b + (a MOD b) = a.

Target variable is the first operand of the infix operation. Result is stored in Target Variable.

Provision of use:

- The *Target* object shall be an active IntegerVariable object.

Syntax description:

```
Modulo                  -->     Target,
                                Value
Target                  -->     GenericObjectReference
Value                   -->     GenericInteger
```

23.5 Formal description

```
IntegerVariable Class       --> Variable Class
```

24 OctetStringVariable Class

Description	Defines a variable of type OctetString within the context of a Group object.
Base class	Variable
Subclasses	None
Status	Concrete class

24.1 Attributes

This subclause defines inherited, exchanged and internal attributes for this class.

24.1.1 Inherited attributes

This class has all the attributes of its base class, with no additional constraints:

24.1.2 Own exchanged attributes

OriginalValue	The OriginalValue attribute shall be an OctetString.

24.1.3 Own internal attributes

This class defines the following additional internal attributes:

Value	• The *Value* attribute shall be an OctetString.

24.2 Events

This class has the same events as its base class, with identical semantics.

24.3 Internal behaviours

This class has the same internal behaviours as its base class, with identical semantics.

24.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base class, with identical semantics. In addition, the following applicable MHEG-5 actions are defined:

SetVariable (*NewVariableValue*)	Provisions of use:
	• The *NewVariableValue* shall be of one of the following types: Integer or OctetString.
	  • When the *NewVariableValue* is of type OctetString, the value of the Target variable is set to the value of *NewVariableValue.*
	  • When the *NewVariableValue* is of type Integer, it is first converted to an OctetString, then value of the Target variable is set to that OctetString value.
	  Rule for conversion:
	  1. Integer conversion to OctetString shall make use of the Application CharacterSet attribute.

2. Conversion shall be made to the representation of the integer in base 10.

Append (*AppendValue*) — Appends *AppendValue* to *Target* Variable. Target variable is the first operand of the infix operation. Result is stored in Target Variable.

Provision of use:

- The *Target* object shall be an active OctetStringVariable object.

Syntax description:

```
Append                  -->   Target,
                              AppendValue

Target                  -->   GenericObjectReference
AppendValue             -->   GenericOctetString
```

24.5 Formal description

```
OctetStringVariable Class       --> Variable Class
```

25 ObjectRefVariable Class

Description	Defines a variable of type ObjectReference within the context of a Group object.
Base class	Variable
Subclasses	None
Status	Concrete class

25.1 Attributes

This subclause defines inherited, exchanged and internal attributes for this class.

25.1.1 Inherited attributes

This class has all the attributes of its base class, with no additional constraints:

25.1.2 Own exchanged attributes

OriginalValue	The OriginalValue attribute shall be an ObjectReference.

25.1.3 Own internal attributes

This class defines the following additional internal attributes:

Value	• The *Value* attribute shall be an ObjectReference.

25.2 Events

This class has the same events as its base class, with identical semantics.

25.3 Internal behaviours

The internal behaviours of this class are the same semantics as for its base class.

25.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base class, with identical semantics. In addition, the following applicable MHEG-5 actions are defined:

SetVariable (*NewVariableValue*)	Provisions of use: • The *NewVariableValue* shall be a GenericObjectReference.

25.5 Formal description

```
ObjectRefVariable Class      --> Variable Class
```

26 ContentRefVariable Class

Description	Defines a variable of type ContentReference within the context of a Group object.
Base class	Variable
Subclasses	None
Status	Concrete class

26.1 Attributes

This subclause defines inherited, exchanged and internal attributes for this class.

26.1.1 Inherited attributes

This class has all the attributes of its base class, with no additional constraints:

26.1.2 Own exchanged attributes

OriginalValue	The OriginalValue attribute shall be a ContentReference.

26.1.3 Own internal attributes

This class defines the following additional internal attributes:

Value	• The *Value* attribute shall be a ContentReference.

26.2 Events

This class has the same events as its base class, with identical semantics.

26.3 Internal behaviours

The internal behaviours of this class are the same semantics as for its base class.

26.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base class, with identical semantics. In addition, the following applicable MHEG-5 actions are defined:

SetVariable (*NewVariableValue*)	Provisions of use: • The *NewVariableValue* shall be a GenericContentReference.

26.5 Formal description

```
ContentRefVariable Class      --> Variable Class
```

27 Presentable Class

Description	Defines the behaviour of objects that may be presented within a Scene.
Base class	Ingredient
Subclasses	Stream, Visible, Audio, TokenGroup
Status	Abstract class

27.1 Attributes

This subclause defines the inherited, exchanged and internal attributes for this class.

27.1.1 Inherited attributes

This class has all the attributes of its base class, with identical semantics.

27.1.2 Own exchanged attributes

This class defines no additional exchanged attribute.

27.1.3 Own internal attributes

This class defines no additional internal attribute.

27.2 Events

This class has the same events as its base class, with identical semantics.

27.3 Internal behaviours

The internal behaviours of this class have the same semantics as for its base class.

27.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base class, with identical semantics, except as noted below (SetData). In addition, the following applicable MHEG-5 actions are defined:

SetData Execute synchronously the following actions:

1. Execute the SetData action as defined in the Ingredient class.
2. If the Presentable is active, present it again immediately using the new value of *Content*.

Provisions of use and syntax description are provided in the Ingredient class.

Run Execute the following actions:

1. If the target Presentable is active, disregard this action.
2. If the target Presentable is non-available or is inactive, apply the *Activation* behaviour of the Presentable.

Syntax description:

```
Run                    -->  Target
```

```
Target                    -->  GenericObjectReference
```

Stop Execute the following actions:

1. If the target Presentable is inactive, disregard this action.
2. If the target Presentable is active, apply the *Deactivation* behaviour of the Presentable.

Provisions of use:

- The *Target* object shall be available.

Syntax description:

```
Stop                      -->  Target
Target                    -->  GenericObjectReference
```

27.5 Formal description

```
Presentable Class         -->  Ingredient Class
```

28 TokenManager Class

Description	Mix-in class that defines functions to manage the navigation of a logical token among a group of elements. The token may be used to give a special behaviour to one element in the group, such as the highlight in a jumping-highlight navigation scheme.
Base class	None (mix-in)
Subclasses	TokenGroup
Status	Abstract class

28.1 Attributes

This subclause defines inherited, exchanged and internal attributes for this class.

28.1.1 Inherited attributes

This class has no inherited attributes.

28.1.2 Own exchanged attributes

This class defines the following exchanged attributes:

MovementTable — Table that indicates to which element to move the token according to the previous element and an arbitrary number of movements.

The MovementTable attribute describes a discrete function *c2 = f(c1, m)*, where *c1* and *c2* are one-based indices of elements, and *m* is the *Movement* parameter used in the definition of the Move action. The index 0 indicates that no element holds the token.

For instance, if the token is on element 2, and the action Move(4) is executed, the expression *f(2, 4)* evaluates to the number of the element to get the token.

The function *f* is represented as a *N x M* array, where *N* is the number of elements in the group and M is the number of possible movements. The elements are referred to by a one-based numeric index.

- Optional attribute.
- Sequence of Movement data structures. Each Movement has an implicit MovementIdentifier, which is a one-based Integer index computed according to the position of the Movement in the sequence. Each Movement further consists of:
  - TargetElements: Sequence of Integers. These Integers define which element will get the token when a Movement is performed.
- Default value: none

Movement Table Example:

Appearance of elements on the screen:

Element 1 Element 2

Element 3 Element 4

Content of the Movement Table:

Token at:	Element 1	Element 2	Element 3	Element 4

Move(1)	1	2	1	2
Move(2)	3	4	3	4
Move(3)	1	1	3	3
Move(4)	2	2	4	4
Move(5)	4	2	3	4
Move(6)	1	2	2	4

According to this table, the action Move(3) when the token is on Element2 would result in Element1 getting the token. The action Move(4) when the token is on Element 4 would have no effect at all.

The movements shown in the table could map to remote control keys as follows:

Move(1) -> UpArrow
Move(2) -> DownArrow
Move(3) -> LeftArrow
Move(4) -> RightArrow
Move(5) -> DownDiagonalRight
Move(6) -> UpDiagonalRight

NOTE - If a cell in the MovementTable contains the value 0, the corresponding Move will result in no element having the token. See *TokenPosition* below.

28.1.3 Own internal attributes

This class defines the following additional internal attributes:

TokenPosition Index of the element that currently has the token.

- Integer within the range [0, number of elements].
- Initial value: 1.
- The value 0 has the special meaning that no element has the token.

28.2 Events

This class defines the following events:

TokenMovedFrom This event is generated when an element loses the token.

- Associated data: Index. This is an Integer representing the index of the element that loses the token.

TokenMovedTo This event is generated when an element gets the token.

- Associated data: Index. This is an Integer representing the index of the element that gets the token.

28.3 Internal behaviours

This class defines the following internal behaviour:

TransferToken (TargetElement) Execute the following sequence of actions:

1. Generate a *TokenMovedFrom* event with the value of the *TokenPosition* attribute as associated data.
2. Set the *TokenPosition* attribute to the value of the *TargetElement* parameter..

3. Generate a *TokenMovedTo* event with the value of the *TokenPosition* attribute as associated data.

   The *TokenMovedTo* and *TokenMovedFrom* events will be generated even if the value of *TokenPosition* after (before) the token moved was 0.

28.4 Effect of MHEG-5 actions

This class defines the following applicable MHEG-5 action:

Move (*Movement Identifier*)

Move the token between elements of the group. The movement to apply from any particular element location is described in the *MovementTable* attribute.

Execute the following sequence of actions:

1. Determine the *TargetElement* by using the MovementTable.
2. If the *TargetElement* does not have the token yet, apply the *Transfer-Token(TargetElement)* behaviour of the TokenManager.

Provisions of use:

- The *Target* object shall be an available TokenManager.

Syntax description:

```
Move                  -->   Target,
                            Movement Identifier
Target                -->   GenericObjectReference
Movement Identifier   -->   GenericInteger
```

MoveTo (*Index*)

Move the token to a specific element of the group.

Execute the following sequence of actions:

1. Determine the *TargetElement* by using the *Index* parameter of the MoveTo action.
2. If the target element does not have the token yet, apply the *Transfer-Token(TargetElement)* behaviour of the TokenManager.

Provisions of use:

- The *Target* object shall be an available TokenManager.
- *Index* shall be set within the range [0, *N*], where *N* is the number of elements in the group.

Syntax description:

```
MoveTo                --->  Target,
                            Index
Target                --->  GenericObjectReference
Index                 --->  GenericInteger
```

GetTokenPosition (*TokenPositionVar*)

Set the Variable referenced by *TokenPositionVar* to the value of the *TokenPosition* attribute.

Provisions of use:

- The *Target* object shall be an available TokenManager.
- *TokenPositionVar* shall refer to an active IntegerVariable object.

Syntax description:

```
GetTokenPosition      -->   Target,
                            TokenPositionVar
Target                -->   GenericObjectReference
TokenPositionVar      -->   ObjectReference
```

28.5 Formal description

```
TokenManager Class      --> MovementTable?
MovementTable           --> Movement+
Movement                --> TargetElement+
TargetElement           --> INTEGER
```

29 TokenGroup Class

Description	Defines a group of Visible objects for navigation. Each Visible object in the group may hold a token, as defined in the TokenManager class. On the basis of the events that are generated when that token moves, a special behaviour for the Visible that holds the token may be implemented, such as a highlight.
Base class	Presentable, TokenManager
Subclasses	ListGroup
Status	Concrete class

29.1 Attributes

This subclause defines inherited, exchanged and internal attributes for this class.

29.1.1 Inherited attributes

This class has all the attributes of its base class, with the following constraints:

Attribute Name	Defined in	Constraints and Requirements
ContentHook	Ingredient	This attribute shall not be encoded for this class.
OriginalContent	Ingredient	This attribute shall not be encoded for this class.
TokenPosition	TokenManager	In the TokenGroup class, this attribute shall take values only in the range *[0,N]*, where *N* is the number of Visibles in the TokenGroup. The value 0 signifies that no Visible has the token.

29.1.2 Own exchanged attributes

This class defines the following additional exchanged attributes:

TokenGroupItems — Set of TokenGroupItems that belong to the group. Each TokenGroupItem contains a reference to a Visible and a number of Actions, called ActionSlots. These actions may be executed on demand by the CallActionSlot action.

- Sequence of the following data structures:
  - AVisible: reference to a Visible object.
  - ActionSlots: Optional sequence of Action objects.

NoTokenActionSlots — Set of ActionSlot that may be executed on demand by the CallActionSlot action when no item has the token.

- Optional attribute.
- Sequence of ActionSlot.

29.1.3 Own internal attributes

This class defines no additional internal attribute.

29.2 Events

This class has the same events as its base classes, with identical semantics.

29.3 Internal behaviours

The following internal behaviours of this class have changes in semantics:

Activation Execute the following sequence of actions:

1. Apply the *Activation* behaviour as inherited from the Presentable class.
2. Apply the activation behaviour to each item in the order given by TokenGroupItems.
3. Generate a *TokenMovedTo* event with the value of the *TokenPosition* attribute as associated data.
4. Set the *RunningStatus* to True and generate an *IsRunning* event.

Deactivation

1. Generate a TokenMoveFrom event with the value of the *TokenPosition* attribute as associated data.
2. Apply the *Deactivation* behaviour as inherited from the Presentable class.

29.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base class, with identical semantics, except as defined below. In addition, the following applicable MHEG-5 actions are defined:

CallActionSlot (*Index*) Execute an Action object associated with the item that currently has the token.

Execute the following sequence of actions:

1. If *TokenPosition* is set to 0, consider the ActionSlots sequence of the NoTokenActionSlots attribute.
2. If *TokenPosition* is different from 0, consider the ActionSlots sequence that is associated with the item that currently has the token.
3. Consider the ActionSlot that is at position *Index* in the ActionSlots sequence. If this ActionSlot is *NULL*, then ignore the effect of the CallActionSlot action. Otherwise, execute that ActionSlot.

The execution of the CallActionSlot action is not completed until the ActionSlot action has been executed.

Provisions of use:

- The *Target* object shall be an active TokenGroup object.
- *Index* shall be set in the range [0, number of ActionSlots associated with the item that currently has the token].

Syntax description:

```
CallActionSlot      -->   Target,
                          Index
Target              -->   GenericObjectReference
Index               -->   GenericInteger
```

29.5 Formal description

```
TokenGroup Class        -->  Presentable Class,
                             TokenManager Class,
                             TokenGroupItems,
                             NoTokenActionSlots?
NoTokenActionSlots      -->  ActionSlots
TokenGroupItems         -->  TokenGroupItem+
TokenGroupItem          -->  AVisible,
                             ActionSlots?
AVisible                -->  ObjectReference
ActionSlots             -->  ActionSlot+
ActionSlot              -->  Action Class | NULL
```

30 ListGroup Class

Description	This class defines locations on the screen for each position as defined in the *MovementTable* inherited from the base class. Furthermore it defines functionality to present Visibles contained in an internal list at these locations and to add Visibles to and remove Visibles from this internal list.
Base class	TokenGroup
Subclasses	None
Status	Concrete class

NOTES

1 SetPosition, GetPosition, Run, Stop and other elementary actions may be targeted to Visibles in ListGroup, however the author should be warned of the possible side effects of such actions (e.g. regarding *Position* of Visibles).

2 Although not recommended, it is allowed to have a Visible referenced by several ListGroups, in such case, behaviours of that Visible depend on which ListGroup or elementary actions are invoked. The author should be aware of possible side effects.

30.1 Attributes

This subclause defines inherited, exchanged and internal attributes for this class.

30.1.1 Inherited attributes

This class has all the attributes of its base class.

30.1.2 Own exchanged attributes

This class defines the following additional exchanged attributes:

Positions — Set of screen co-ordinates associated with the positions as specified by the *TokenMovementTable* attribute inherited from TokenGroup. Together these attributes specify the *cells* of the ListGroup. Each cell has an index, identical to the logical position as defined in the *TokenMovementTable*.

- Sequence of the following data structure:
  - Position: pair of integers (XPosition, YPosition).

WrapAround — An optional Boolean indicating the behaviour of the ListGroup with respect to the presentation of the items in the *ItemList*.

- Optional Boolean.
- Default value: False.

MultipleSelection — An optional Boolean that determines if the ListGroup allows multiple items to have their *ItemSelectionStatus* set to True simultaneously.

- Optional Boolean.
- Default value: False.

30.1.3 Own internal attributes

This class defines the following additional internal attributes:

ItemList Set of references to Visibles that belong to the group. Each reference has implicitly an index number, starting from 1. This index number is used to refer to the references in the *ItemList* attribute.

Each item in the *ItemList* has a *ItemSelectionStatus* attribute. This *ItemSelectionStatus* is a Boolean, with a default value of False. This attribute can be set using the *SelectItem* and *DeselectItem* elementary actions; its value can be returned using the *GetItemStatus* elementary action. Note that this attribute is not part of the Visible objects referenced from the *ItemList*.

When the ListGroup is active, items of the *ItemList*, starting with the item which is indicated by the *FirstItem* attribute, are presented at the cells. The other items are made inactive. The presentation depends also on the value of the WrapAround attribute.

At preparation, this set is initialised to the set of references listed in the *TokenGroupItems* attribute.

- Set of references to Visibles
  - Initial value: Empty set

FirstItem The index of the item of the *ItemList* which is presented at the first cell. This defines a 'window' on the ordered list of items in the *ItemList*. This window is equal in size to the number of cells and the position of this window with respect to the items can be changed using the *ScrollItems* elementary action.

The presentation of the items in the list depends on the position of this window, and the value of the *WrapAround* attribute.

If there are at least as many items in the *ItemList* as there are cells, the window can be visualised easily. This situation is depicted twice below. In the first example, *WrapAround* is false, i.e. the list is not wrapping:

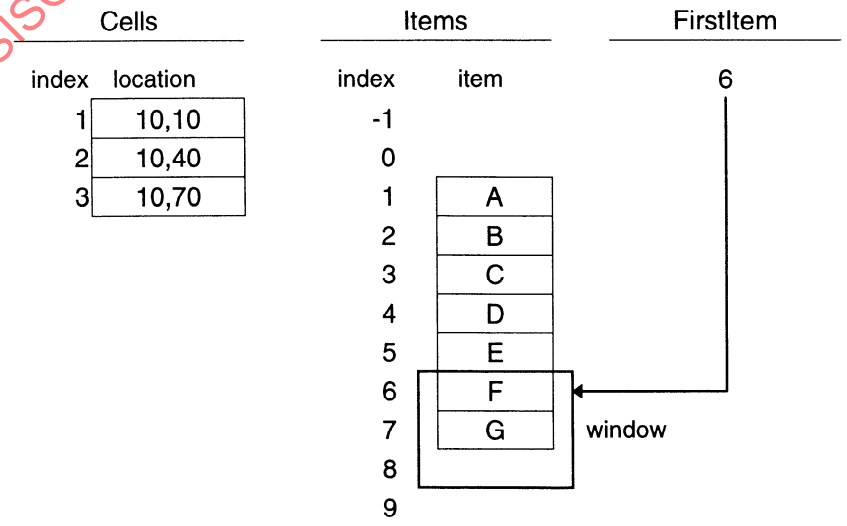


Figure 6 - Presentation when WrapAround is False

In this case, items F and G are presented at cell 1 and 2 respectively, while the other items are not presented. Cell 3 remains unused.

In the next example, WrapAround is true, i.e. the list is wrapping:

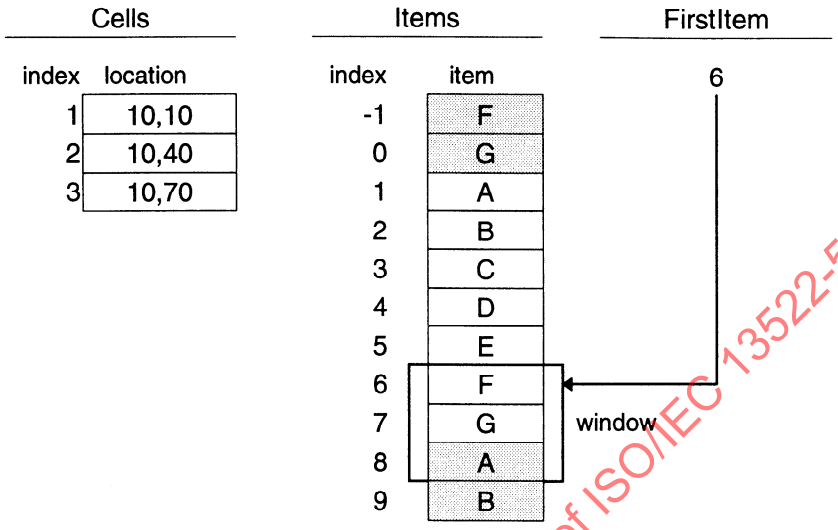


Figure 7 - Presentation when WrapAround is True

In this case, items F, G and A are presented at cell 1, 2 and 3 respectively, while the other items are not presented. Note that in a wrapping list, the indices of the items are extended (modulo the number of items in the list). For instance, ...,-4, 3, 10,... are all valid indices for item C in the above example.

The other situation occurs if there are less items in the ItemList as there are cells. This situation is depicted twice below. In this next example, *WrapAround* is false, i.e. the list is not wrapping:

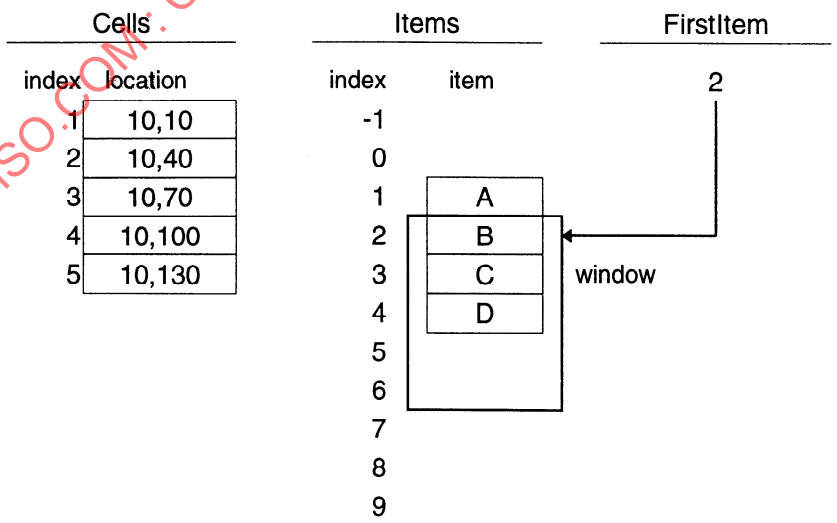


Figure 8 - Presentation of few elements when WrapAround is False

This case is much like the first example only the window is bigger. Items B, C and D are presented at cell 1, 2 and 3 respectively, while item A is not presented. Cells 4 and 5 remain unused.

In the next example, *WrapAround* is true, i.e. the list is wrapping:

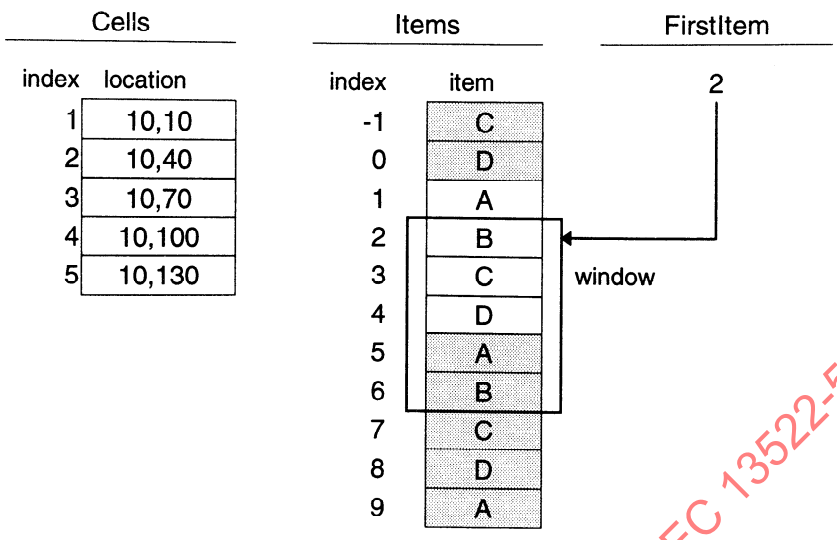


Figure 9 - Presentation of few elements when WrapAround is True

The important thing in this case is that there are two candidate cells to present item B. Since Visibles can have just one position, items can also just be presented in one cell. If according to the window an item can be presented at more than one cell, it will be presented only at the cell with the lowest index. In this case, item B will be presented at cell 1. Items C,D and A are presented at cell 2, 3 and 4 respectively. Cell 5 remains unused. Note that when the number of items is less than the number of cells, cells will remain unused, even if the list is wrapping.

Finally, any Integer value is allowed for FirstItem. In some cases, when WrapAround is False, it may occur that every cell is empty; such a case, where no Visible is displayed in the window, is allowed.

- Integer.
  - Initial value: 1.

30.2 Events

This class has the same events as its base classes, with identical semantics. In addition, the following events are defined:

FirstItemPresented

This event is generated each time the presentation status of the first item in the *ItemList* changes. The associated value will reflect the new presentation status. The presentation status of the item can change if the *FirstItem* attribute is changed, or if the number of items in the list is changed.

- Associated data: Boolean. True if the item is presented, False if it is not presented.

LastItemPresented

This event is generated each time the presentation status of the last item in the *ItemList* changes. The associated value will reflect the new presentation status. The presentation status of the item can change if the *FirstItem* attribute is changed, or if the number of items in the list is changed.

- Associated data: Boolean. True if the item is presented, False if it is not presented.

HeadItems — This event is generated each time the number of items in the *ItemList* with an index in the range [1,*FirstItem*-1] changes. This number can change if the *FirstItem* attribute is changed, or if the number of items in the list is changed.

- Associated data: Integer - the number of items in the *ItemList* with an index smaller than *FirstItem*.

TailItems — This event is generated each time the number of items in the *ItemList* with an index in the range [*FirstItem*, number of items] changes. This number can change if the *FirstItem* attribute is changed, or if the number of items in the list is changed.

- Associated data: Integer - the number of items in the *ItemList* with an index greater than or equal to *FirstItem*.

ItemSelected — This event is generated if the *ItemSelectionStatus* of an item changes to True.

- Associated data: Integer - the index of the item.

ItemDeselected — This event is generated if the *ItemSelectionStatus* of an item changes to False.

- Associated data: Integer - the index of the item.

30.3 Internal behaviours

This class defines the following internal behaviours:

Preparation — Execute the following sequence of actions:

1. Apply the *Preparation* behaviour as inherited from the base class.
2. Add each reference listed in the *TokenGroupItems* attribute to the *ItemList*. If a Visible is referenced more than once in the *TokenGroupItems*, it is added only once in the *ItemList*.

Destruction — Execute the following sequence of actions:

1. Reset all Visibles of the ListGroup to their OriginalPosition.
2. Apply the *Destruction* behaviour as inherited from the base class.

Activation — Execute the following sequence of actions:

1. Apply the *Activation* behaviour as inherited from the base class.
2. Apply the *Update* behaviour.

Deactivation — Execute the following sequence of actions:

1. Apply the *Deactivation* behaviour to all Visibles referenced in the *ItemList*.
2. Apply the *Deactivation* behaviour as defined in the base class.

Update — Execute the following sequence of actions:

1. For each item to be presented at a certain cell, set its *Position* (internal attribute) to the position defined for that cell. Subsequently, if the *RunningStatus* of the ListGroup is true, and the item is inactive, apply the *Activation* behaviour to it.
2. For each item not to be presented set its position to its OriginalPosition attribute. Subsequently, if the *RunningStatus* of the ListGroup is true, and the item is active, apply the *Deactivation* behaviour to it.

Additem (Index, Item)

Execute the following sequence of actions:

1. If the Visible referenced by the *Item* parameter is already in the *ItemList*, ignore this behaviour.
2. If *Index* is less than 1 or greater than the current number of ItemList + 1, ignore this behaviour.
3. Add the Item reference at position *Index* in the *ItemList*. (The item previously at position *Index* in the ItemList -if any- will now have index *Index+1* and, similarly, every item of index greater than *Index* will now have an index incremented by one.)
4. Apply the *Update* behaviour.

Delitem (Item)

Execute the following sequence of actions:

1. If the Visible referenced by the *Item* parameter is not in the *ItemList*, ignore this action.
2. Remove the reference indicated by the *Item* parameter from the *ItemList*. (The item index of the following items in ItemList will decrement by one.)
3. Set the position of the referenced Visible to its *OriginalPosition*.
4. Apply the *Update* behaviour.

Select (ItemIndex)

Execute the following sequence of actions:

1. If *ItemSelectionStatus* of item with index *ItemIndex* is True, ignore this behaviour.
2. If *MultipleSelection* is False, apply *Deselect(Index)* internal behaviour for any item with index *Index*, for which the *ItemSelectionStatus* is True.
3. Set the *ItemSelectionStatus* of the item with index *ItemIndex* to True.
4. Generate an *ItemSelected* event with *ItemIndex* as associated data.

Deselect (ItemIndex)

Execute the following sequence of actions:

1. If *ItemSelectionStatus* of item with index *ItemIndex* is False, ignore this behaviour.
2. Set the *ItemSelectionStatus* of the item with index *Index* to False.
3. Generate an *ItemDeselected* event with *Index* as associated data.

30.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base class, with identical semantics. In addition, the following applicable MHEG-5 actions are defined:

AddItem (*ItemIndex*, *VisibleReference*)

Apply the *Additem(ItemIndex, VisibleReference)* internal behaviour to add the reference to the visible, as indicated by the *VisibleReference* parameter at the position indicated by the *ItemIndex* parameter.

Provisions of use:

- The *Target* object shall be an available ListGroup object.
- *VisibleReference* shall refer to an available Visible object, which is not referred to by any reference in the *ItemList*.

Syntax description:

```
AddItem              -->    Target,
                            ItemIndex,
                            VisibleReference
Target               -->    GenericObjectReference
ItemIndex            -->    GenericInteger
VisibleReference     -->    GenericObjectReference
```

DelItem (*VisibleReference*)

Apply the *Delitem(VisibleReference)* internal behaviour to delete the reference to the visible, as indicated by the *VisibleReference* parameter from the ItemList attribute, if it occurs in it.

Provisions of use:

- The *Target* object shall be an available ListGroup object

Syntax description:

```
DelItem              -->    Target,
                            VisibleReference
Target               -->    GenericObjectReference
VisibleReference     -->    GenericObjectReference
```

GetListItem (*ItemIndex, ItemRefVar*)

Return the reference included in the *ItemList* attribute with the index specified by the *ItemIndex* parameter in the ObjectRefVariable referenced by *ItemRefVar*.

If *WrapAround* is False, and if the *ItemIndex* specifies an index smaller than 1 or greater than the number of items in the *ItemList* attribute, ignore the action.

If *WrapAround* is True, and if the *ItemIndex* specifies an index smaller than 1 or greater than the number of items in the *ItemList* attribute, the *ItemIndex* shall be interpreted modulo the number of items in the *ItemList* attribute.

Provisions of use:

- The *Target* object shall be an available ListGroup object.
- *ItemRefVar* shall refer to an active ObjectRefVariable object.
- If WrapAround is False, *ItemIndex* shall be in the range [1, Number of Items].

Syntax description:

```
GetListItem          -->    Target,
                            ItemIndex,
                            ItemRefVar
```

```
Target              -->    GenericObjectReference
ItemIndex           -->    GenericInteger
ItemRefVar          -->    ObjectReference
```

GetCellItem (*CellIndex, ItemRefVar*)

Return the reference of the visible presented at the cell indicated by the *CellIndex* parameter in the ObjectRefVariable referenced by *ItemRefVar*.

If the *CellIndex* specifies an index smaller than or equal to 1 return the reference of the visible presented at the first cell. If the *CellIndex* specifies an index greater than or equal to the number of cells, return the reference of the visible presented in the last cell. If no visible is presented at the indicated cell, return *NULL*.

NOTE - the GetTokenPosition action can be used to get the index of the cell which currently holds the token, as defined in the base class.

Provisions of use:

- The *Target* object shall be an available ListGroup object.
- *ItemRefVar* shall refer to an active ObjectRefVariable object.

Syntax description:

```
GetCellItem         -->    Target,
                           CellIndex,
                           ItemRefVar
Target              -->    GenericObjectReference
CellIndex           -->    GenericInteger
ItemRefVar          -->    ObjectReference
```

GetItemStatus (*ItemIndex, ItemStatusVar*)

Return the value of the *ItemSelectionStatus* attribute of the item in the *ItemList* with index *ItemIndex* in the BooleanVariable referenced by *ItemStatusVar*.

If *WrapAround* is False, and if the *ItemIndex* specifies an index smaller than 1 or greater than the number of items in the ItemList attribute, ignore the action.

If *WrapAround* is True, and if the *ItemIndex* specifies an index smaller than 1 or greater than the number of items in the *ItemList* attribute, the *ItemIndex* shall be interpreted modulo the number of items in the *ItemList* attribute.

Provisions of use:

- The *Target* object shall be an available ListGroup object.
- *ItemRefVar* shall refer to an active BooleanVariable object.
- If WrapAround is False, *ItemIndex* shall be in the range [1, Number of Items].

Syntax description:

```
GetItemStatus       -->    Target,
                           ItemIndex,
                           ItemStatusVar
Target              -->    GenericObjectReference
```

```
ItemIndex              -->    GenericInteger
ItemStatusVar          -->    ObjectReference
```

SelectItem (*ItemIndex*)

If *WrapAround* is False, and if the *ItemIndex* specifies an index smaller than 1 or greater than the number of items in the *ItemList* attribute, ignore this action, otherwise:

If *WrapAround* is True, and if the *ItemIndex* specifies an index smaller than 1 or greater than the number of items in the *ItemList* attribute, the *ItemIndex* shall be interpreted modulo the number of items in the *ItemList* attribute.

Apply the *Select(ItemIndex)* internal behaviour.

Provisions of use:

- The *Target* object shall be an available ListGroup object

Syntax description:

```
SelectItem             -->    Target,
                              ItemIndex
Target                 -->    GenericObjectReference
ItemIndex              -->    GenericInteger
```

DeselectItem (*ItemIndex*)

If *WrapAround* is False, and if the *ItemIndex* specifies an index smaller than 1 or greater than the number of items in the *ItemList* attribute, ignore this action, otherwise:

If *WrapAround* is True, and if the *ItemIndex* specifies an index smaller than 1 or greater than the number of items in the *ItemList* attribute, the *ItemIndex* shall be interpreted modulo the number of items in the *ItemList* attribute.

Apply the *Deselect(ItemIndex)* internal behaviour.

Provisions of use:

- The *Target* object shall be an available ListGroup object

Syntax description:

```
DeselectItem           -->    Target,
                              ItemIndex
Target                 -->    GenericObjectReference
ItemIndex              -->    GenericInteger
```

ToggleItem (*ItemIndex*)

If *WrapAround* is False, and if the *ItemIndex* specifies an index smaller than 1 or greater than the number of items in the *ItemList* attribute, ignore this action, otherwise:

If *WrapAround* is True, and if the *ItemIndex* specifies an index smaller than 1 or greater than the number of items in the *ItemList* attribute, the *ItemIndex* shall be interpreted modulo the number of items in the *ItemList* attribute.

If the *ItemSelectionStatus* of the item indicated by *ItemIndex* is True, apply the *Deselect(ItemIndex)* internal behaviour, otherwise apply the *Select(ItemIndex)* internal behaviour.

Provisions of use:

- The *Target* object shall be an available ListGroup object

Syntax description:

```
ToggleItem          -->   Target,
                          ItemIndex,
Target              -->   GenericObjectReference
ItemIndex           -->   GenericInteger
```

ScrollItems (*ItemsToScroll*)

Add *ItemsToScroll* to the *FirstItem* attribute, and apply the *Update* behaviour.

Provisions of use:

- The *Target* object shall be an available ListGroup object

Syntax description:

```
ScrollItems         -->   Target,
                          ItemsToScroll
Target              -->   GenericObjectReference
ItemsToScroll       -->   GenericInteger
```

SetFirstItem (*NewFirstItem*)

Set the value of the *FirstItem* attribute to *NewFirstItem*. Apply the Update behaviour.

Provisions of use:

- The *Target* object shall be an available ListGroup object

Syntax description:

```
SetFirstItem        -->   Target,
                          NewFirstItem
Target              -->   GenericObjectReference
NewFirstItem        -->   GenericInteger
```

GetFirstItem (*FirstItemVar*)

Return the current value of the *FirstItem* attribute in the IntegerVariable referenced by *FirstItemVar*.

Provisions of use:

- The *Target* object shall be an available ListGroup object.
- *FirstItemVar* shall refer to an active IntegerVariable object.

Syntax description:

```
GetFirstItem          -->    Target,
                             FirstItemVar
Target                -->    GenericObjectReference
FirstItemVar          -->    ObjectReference
```

GetListSize (*SizeVar*)

Return the number of items in the *ItemList* in the IntegerVariable referenced by *SizeVar*.

Provisions of use:

- The *Target* object shall be an available ListGroup object.
- *SizeVar* shall refer to an active IntegerVariable object.

Syntax description:

```
GetListSize           -->    Target,
                             SizeVar
Target                -->    GenericObjectReference
SizeVar               -->    ObjectReference
```

30.5 Formal description

```
ListGroup Class       -->    TokenGroup Class,
                             Positions,
                             WrapAround?,
                             MultipleSelection?
Positions             -->    Position*
Position              -->    XYposition,
WrapAround            -->    BOOLEAN
MultipleSelection     -->    BOOLEAN
```

31 Visible Class

Description	Defines the behaviour of Presentables that have a visual representation on the screen.
Base class	Presentable
Subclasses	Video, RTGraphics, Bitmap, LineArt, Text, Slider, Button
Status	Abstract class

31.1 Attributes

This subclause defines the inherited, exchanged and internal attributes for this class.

31.1.1 Inherited attributes

This class has all the attributes of its base class, with identical semantics.

31.1.2 Own exchanged attributes

This class defines the following additional exchanged attributes:

OriginalBoxSize

Original size of the bounding box of the Visible object with respect to the coordinate system of the Scene. Size is given by a pair of Integers representing respectively the width (length along X-axis) and height (length along Y-axis) of the box.

Negative values shall not be used. Values exceeding the scene coordinate system are allowed; in that case only the part of the Visible within the scene coordinate system shall be rendered.

- Pair of Integers (XBoxSize, YBoxSize).

OriginalPosition

Original position of the top left corner of the Visible with respect to the coordinate system of the Scene.

This is a set of (X, Y) co-ordinates expressing the original position of the top left corner of the visible in the coordinate system of the currently active Scene.

Negative values and values exceeding the scene coordinate system are allowed; in that case only the part of the Visible within the scene coordinate system shall be rendered.

- Optional pair of Integers (XPosition, YPosition).
- Default value: (0,0).

OriginalPaletteRef

Indicate the initial Palette object that shall be used to render colours of the Visible object.

- Optional reference to a Palette object.
- Default value: No palette.

This attribute is mandatory when other colour attributes are set and need Palette references.

31.1.3 Own internal attributes

This class defines the following additional internal attributes:

BoxSize Size of the bounding box of the Visible object with respect to the coordinate system of the Scene. Parts of the Visible that fall outside the borders of the bounding box shall not be rendered. If the Visible does not completely cover the contents of the bounding box, then the remaining parts of the bounding box shall be transparent.

- Pair of Integers (XBoxSize, YBoxSize).
- Initial value: Value of the OriginalBoxSize attribute.

Position Position of the top left corner of the Visible with respect to the coordinate system of the Scene. The Visible, when rendered, shall have its top left corner at this position.

- Pair of Integers (XPosition, YPosition).
- Initial value: Value of the OriginalPosition attribute.

PaletteRef If a Palette is used, reference to the Palette object that shall be used to render colours of the Visible object.

The *Palette* attribute shall not be defined for classes specifying that OriginalPaletteRef shall not be encoded.

- Optional reference to a Palette object.
- Initial value: Value of the OriginalPaletteRef attribute.

31.2 Events

This class has the same events as its base class, with identical semantics.

31.3 Internal behaviours

This class has the same internal behaviours as its base class, with the following changes in semantics:

Preparation Execute the following sequence of actions:

1. Execute steps 1, 2 and 3 of the *Preparation* behaviour as defined in the Root class.
2. If the Visible object is not referenced in the *DisplayStack* of the active Application object, add a reference to this Visible at the top of the *DisplayStack*.
3. Execute steps 4, 5 and 6 of the *Preparation* behaviour as defined in the Root class.

Destruction Execute the following sequence of actions:

1. If the Visible object is referenced in the *DisplayStack* of the active Application object, remove the reference to this Visible from the *DisplayStack*.
2. Execute the *Destruction* behaviour as defined in the base class.

Activation Execute the following sequence of actions:

1. Execute the *Activation* behaviour as defined in the base class.
2. Display the Visible according to its position in the *DisplayStack* and to the position and the bounding box defined by the *Position* and *BoxSize* attributes.

3. Set the *RunningStatus* to True and generate an *IsRunning* event.

Deactivation

Execute the following sequence of actions:

1. If the RunningStatus attribute of the object is False, ignore the behaviour, otherwise:
2. Stop displaying the Visible.
3. Execute *Deactivation* behaviour as defined in the base class.

31.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base class, with identical semantics. In addition, the following applicable MHEG-5 actions are defined:

SetPosition (*NewXPosition*, *NewYPosition*)

Change the location of the target Visible.

Execute the following sequence of actions:

1. Set the *Position* attribute according to *NewXPosition* and *NewYPosition*.
2. If the Visible object is active, redraw the graphic widget representing the object on the screen in the bounding box defined by the *BoxSize* and *Position* attributes and according to its position in the *DisplayStack* of the active Application object.

Provisions of use:

- The *Target* object shall be an available Visible object.
- *NewXPosition* and *NewYPosition* shall correspond to a location interpreted in the Scene coordinate system defined by the SceneCoordinateSystem attribute of the currently active Scene.

Syntax description:

```
SetPosition          -->   Target,
                           NewXPosition,
                           NewYPosition
Target               -->   GenericObjectReference
NewXPosition         -->   GenericInteger
NewYPosition         -->   GenericInteger
```

GetPosition (*XPositionVar*, *YPositionVar*)

Returns the location of the target Visible.

Set the Variables referenced by *XPositionVar* and *YPositionVar* to the value of the X and Y position of the target visible respectively.

Provisions of use:

- The *Target* object shall be an available Visible object.
- *XPositionVar* and *YPositionVar* shall refer to active IntegerVariable objects.

Syntax description:

```
GetPosition          -->  Target,
                          XPositionVar,
                          YPositionVar
Target               -->  GenericObjectReference
XPositionVar         -->  ObjectReference
YPositionVar         -->  ObjectReference
```

SetBoxSize (*XNewBoxSize*, *YNewBoxSize*)

Change the size of the bounding box of the target Visible.

Execute the following sequence of actions:

1. Set the *BoxSize* attribute.
2. Redraw the graphic widget representing the object on the screen in the bounding box defined by the *BoxSize* and *Position* attributes and according to its position in the *DisplayStack* of the active Application object.

Provisions of use:

- The *Target* object shall be an available Visible object.
- *XNewBoxSize* and *YNewBoxSize* shall be positive values and different from 0.

Syntax description:

```
SetBoxSize           -->  Target,
                          XNewBoxSize,
                          YNewBoxSize
Target               -->  GenericObjectReference
XNewBoxSize          -->  GenericInteger
YNewBoxSize          -->  GenericInteger
```

GetBoxSize (*XBoxSizeVar*, *YBoxSizeVar*)

Returns the size of the bounding box of the target Visible.

Set the Variables referenced by *XBoxSizeVar* and *YBoxSizeVar* to the value of the X and Y position of the target visible respectively.

Provisions of use:

- The *Target* object shall be an available Visible object.
- *XBoxSizeVar* and *YBoxSizeVar* shall refer to active IntegerVariable objects.

Syntax description:

```
GetBoxSize           -->  Target,
                          XBoxSizeVar,
                          YBoxSizeVar
Target               -->  GenericObjectReference
XBoxSizeVar          -->  ObjectReference
YBoxSizeVar          -->  ObjectReference
```

BringToFront

Put a Visible at the foreground of the screen.

Execute the following sequence of actions:

1. If the target Visible object is not referenced in the *DisplayStack* of the active Application object, ignore this action.
2. If the target Visible object is referenced in the *DisplayStack* of the active Application object, move the reference to this Visible at the top of the *DisplayStack*.
3. If the target Visible object is active, redraw the graphic widget representing the object on the screen in the bounding box defined by the *BoxSize* and *Position* attributes and according to its position in the *DisplayStack* of the active Application object.

Provisions of use:

- The *Target* object shall be an available Visible object.

Syntax description:

```
BringToFront        -->   Target
Target              -->   GenericObjectReference
```

SendToBack

Put a Visible at the background of the screen.

Execute the following sequence of actions:

1. If the target Visible object is not referenced in the *DisplayStack* of the active Application object, ignore this action
2. If the target Visible object is referenced in the *DisplayStack* of the active Application object, move the reference to this Visible at the bottom of the *DisplayStack*.
3. If the target Visible object is active, redraw the graphic widget representing the object on the screen in the bounding box defined by the *BoxSize* and *Position* attributes and according to its position in the *DisplayStack* of the active Application object.

Provisions of use:

- The *Target* object shall be an available Visible object.

Syntax description:

```
SendToBack          -->   Target
Target              -->   GenericObjectReference
```

PutBefore (*ReferenceVisible*)

Put a Visible exactly in front of another Visible in the display stack.

Execute the following sequence of actions:

1. If the target Visible object is not referenced in the *DisplayStack* of the active Application object, ignore this action.
2. If the target Visible object is referenced in the *DisplayStack* of the active Application object, move the reference to this Visible at the position exactly before the reference to the *ReferenceVisible* in the *DisplayStack*.
3. If the target Visible object is active, redraw the graphic widget representing the object on the screen in the bounding box defined by the *BoxSize* and *Position* attributes and according to its position in the *DisplayStack* of the

active Application object.

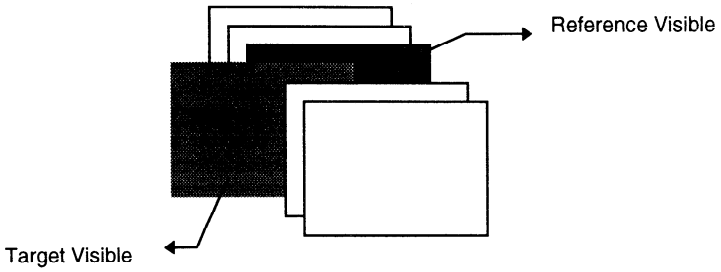


Figure 10 : Effect of PutBefore action

Provisions of use:

- The *Target* object shall be an available Visible object.
- *ReferenceVisible* shall be a reference to an available Visible object.
- The *DisplayStack* of the active Application object shall contain a reference to the *ReferenceVisible* object.

Syntax description:

```
PutBefore              -->   Target,
                             ReferenceVisible

Target                 -->   GenericObjectReference
ReferenceVisible       -->   GenericObjectReference
```

PutBehind (ReferenceVisible)

Put a Visible exactly below another Visible in the display stack.

Execute the following sequence of actions:

1. If the target Visible object is not referenced in the *DisplayStack* of the active Application object, ignore this action.
2. If the target Visible object is referenced in the *DisplayStack* of the active Application object, move the reference to this Visible at the position exactly after the reference to the *ReferenceVisible* in the *DisplayStack*.
3. If the target Visible object is active, redraw the graphic widget representing the object on the screen in the bounding box defined by the *BoxSize* and *Position* attributes and according to its position in the *DisplayStack* of the active Application object.

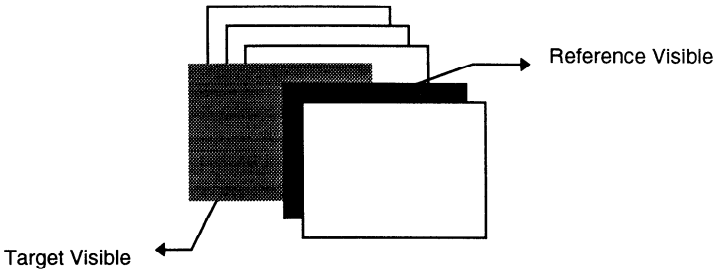


Figure 11 : Effect of PutBehind action

Provisions of use:

- The *Target* object shall be an available Visible object.
- *ReferenceVisible* shall be set or refer to a reference to an available Visible object.
- The *DisplayStack* of the active Application object shall contain a reference to the *ReferenceVisible* object.

Syntax description:

```
PutBehind              -->    Target,
                              ReferenceVisible
Target                 -->    GenericObjectReference
ReferenceVisible       -->    GenericObjectReference
```

SetPaletteRef (*NewPaletteRef*)

Change the colour look-up table used to render colours of the Visible object.

Execute the following sequence of actions:

1. Set *PaletteRef* of the target Visible to *NewPaletteRef*.
2. If the target Visible is active, redraw the Visible by taking into account the new value of *PaletteRef*.

Provisions of use:

- The *Target* object shall be an available Visible object.
- *NewPaletteRef* shall contain a reference to an active Palette object.

Syntax description:

```
SetPaletteRef          -->  Target,
                            NewPaletteRef
Target                 -->  GenericObjectReference
NewPaletteRef          -->  GenericObjectReference
```

31.5 Formal description

```
Visible Class              -->  Presentable class,
                                OriginalBoxSize,
                                OriginalPosition?,
                                OriginalPaletteRef?
OriginalBoxSize            -->  XLength, YLength
XLength                    -->  INTEGER
YLength                    -->  INTEGER
OriginalPosition           -->  XYPosition
OriginalPaletteRef         -->  ObjectReference
```

32 Bitmap Class

Description	Defines the behaviour of a two-dimensional array of pixels.
Base class	Visible
Subclasses	None
Status	Concrete class

32.1 Attributes

This subclause defines the inherited, exchanged and internal attributes for this class.

32.1.1 Inherited attributes

This class has all the attributes of its base class, with the following constraints:

Attribute Name	Defined in	Constraints and Requirements
ContentHook	Ingredient	Either this attribute or its corresponding default attribute in the Application class (BitmapContentHook) is mandatory for this class.
OriginalContent	Ingredient	This attribute is mandatory for this class.

32.1.2 Own exchanged attributes

This class defines the following additional exchanged attributes:

Tiling — When true, this attribute indicates that the bitmap should be replicated (tiled) in the available *BoxSize* of the bitmap.

- Optional Boolean.
- Default value: False.

Original-Transparency — This attribute defines the initial transparency of those pixels of the bitmap that are marked as being transparent.

In cases where the coded representation of the bitmap itself defines the value of the Transparency, the transparency defined by content encoding and the transparency defined by this attribute should be combined. The exact algorithm is not defined by this part of ISO/IEC 13522.

- Optional Integer in the range [0,100] (percent).
- Default value: 0%.

32.1.3 Own internal attributes

This class defines the following additional internal attributes:

Transparency — Defines the transparency of those pixels of the bitmap that are marked as being transparent.

- Optional Integer in the range [0,100] (percent).
- Initial value: Value of the OriginalTransparency attribute.

32.2 Events

This class has the same events as its base class, with identical semantics.

32.3 Internal behaviours

this class has the same behaviours as its base class, with identical semantics.

32.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base class, with identical semantics.

In addition, the following applicable MHEG-5 actions are defined:

ScaleBitmap (*XScale*, *YScale*)

If the MHEG-5 engine implements the Scaling option, the effect of this action is to scale the contents of the Bitmap to the size (*XScale*, *YScale*). Engines that do not implement the scaling option shall ignore this action.

Note that this action does not affect the *BoxSize* internal attribute of the Bitmap object; in other words, the Bitmap is scaled, but its bounding box remains the same.

Provisions of use:

- The *Target* object shall be an available Bitmap object.
- *XScale* and *YScale* shall be positive Integers.

Syntax definition:

```
ScaleBitmap          -->  Target,
                          XScale, YScale
Target               -->  GenericObjectReference
XScale, YScale       -->  GenericInteger
```

SetTransparency (*NewTransparency*)

Execute the following sequence of actions:

1. Change the value of the *Transparency* attribute.
2. If the bitmap is active, redraw it using the new value of the *Transparency* attribute, according to its position in the *DisplayStack* of the active Application object.

Provisions of use:

- The *Target* object shall be an available Bitmap object.
- *NewTransparency* is set within the range [0, 100].

Syntax definition:

```
SetTransparency      -->  Target,
                          NewTransparency
Target               -->  GenericObjectReference
NewTransparency      -->  GenericInteger
```

32.5 Formal description

```
Bitmap Class              -->  Visible Class,
```

```
                                Tiling?,
                                OriginalTransparency?
Tiling                    -->   BOOLEAN
OriginalTransparency      -->   INTEGER
```

33 LineArt Class

Description	Defines functionality associated with vectorial representation of graphical objects.
Base class	Visible
Subclasses	Rectangle, DynamicLineArt
Status	Concrete class

33.1 Attributes

This subclause defines the inherited, exchanged and internal attributes for this class.

33.1.1 Inherited attributes

This class has all the attributes of its base class, with the following constraints:

Attribute Name	Defined in	Constraints and Requirements
ContentHook	Ingredient	Either this attribute or its corresponding default attribute in the Application class (LineArtContentHook) is mandatory for this class.
OriginalContent	Ingredient	This attribute is mandatory for this class.

33.1.2 Own exchanged attributes

This class defines the following additional exchanged attributes:

BorderedBoundingBox

The BorderedBoundingBox attribute determines whether the bounding box - defined by BoxSize and Position attributes - is bordered by lines or not. When this attribute is set to True, all further drawings shall take place inside the border and shall not be drawn on top of it. OriginalLineWidth, OriginalLineStyle and OriginalRefLineColour shall be used for the rendering of the bounding box.

- Optional Boolean attribute.
- Default value: True.

OriginalLineWidth

The OriginalLineWidth attribute determines the original line width of the graphic object including the bounding box.

The OriginalLineWidth attribute is expressed in pixels in the scene coordinate space. It is specified in pixel height for horizontal lines and in pixel width for vertical lines.

The actual rendering resolution and accuracy of the MHEG-5 engine is outside the scope of this part of ISO/IEC 13522.

- Optional Integer attribute.
- Default Value: 1.

OriginalLineStyle

The OriginalLineStyle attribute determines the pattern used for the rendering of the lines of the LineArt object.

- Optional Integer attribute - one of 1, 2 or 3;

  1 means solid, 2 means dashed, 3 means dotted.

- Default value: 1 (solid).

OriginalRefLine-Colour

Initial reference colour for the lines used to render the LineArt object.

The OriginalRefLineColour attribute value is expressed either as a string representing an absolute colour value or as an Integer representing a zero-based index in a colour look-up table. In the latter case, the OriginalPaletteRef attribute must contain a reference to a Palette object that is then used to translate the index to an actual colour value.

The encoding of absolute colour values as well as the actual colour resolution in the rendering process are outside the scope of this part of ISO/IEC 13522.

- Optional attribute.
- Default value: «black».

  The exact appearance on screen of default value «black» is not fully specified by this part of this part of ISO/IEC 13522.
- OctetString or Integer.

OriginalRefFillColour

Initial reference colour for the inside of a closed LineArt object.

The OriginalRefFillColour attribute value is expressed either as a string representing an absolute colour value or as an Integer representing a zero-based index in a colour look-up table. In the latter case, the OriginalPaletteRef attribute must contain a reference to a Palette object that is then used to translate the index to an actual colour value.

If the OriginalRefFillColour is not encoded, a LineArt object that represents a closed shape shall be rendered with a transparent fill colour.

The encoding of absolute colour values as well as the actual colour resolution in the rendering process are outside the scope of this part of ISO/IEC 13522.

- Optional attribute.
- Default value: «transparent».
- OctetString or Integer.

33.1.3 Own internal attributes

This class defines the following additional internal attributes:

LineWidth

The *LineWidth* attribute determines the line width of the graphical objects that are drawn by the graphical actions.

- Integer.
- Initial Value: OriginalLineWidth.

LineStyle

Width of the line that the LineArt object is intended to be rendered with.

- One of 1, 2 or 3;

  1 means solid, 2 means dashed, 3 means dotted.
- Initial Value: OriginalLineStyle.

RefLineColour

Reference colour for the lines used to render the LineArt object.

- OctetString or Integer.
- Initial Value: OriginalRefLineColour.

RefFillColour Reference colour for the background colour used to render the LineArt object.

- OctetString or Integer.
- Initial Value: OriginalRefFillColour.

33.2 Events

This class has the same events as its base class, with identical semantics.

33.3 Internal behaviours

This class defines no additional internal attribute.

33.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base class, with identical semantics. In addition, the following applicable MHEG-5 actions are defined:

SetLineWidth (*NewLineWidth*) Change the width of lines of a LineArt object.

Execute the following sequence of actions:

1. Set the value of the LineWidth attribute to *NewLineWidth*.
2. If the target LineArt object is active, redraw immediately the target object by taking into account the new value of the LineWidth attribute and according to its position in the *DisplayStack* of the active Application object.

Provisions of use:

- The *Target* object shall be an available LineArt object.
- *NewLineWidth* shall be set or refer to a positive integer value.

Syntax description:

```
SetLineWidth        -->   Target,
                          NewLineWidth
Target              -->   GenericObjectReference
NewLineWidth        -->   GenericInteger
```

SetLineStyle (*NewLineStyle*) Change the line style of a LineArt object.

Execute the following sequence of actions:

1. Set the value of the LineStyle attribute to *NewLineStyle*.
2. If the target LineArt object is active, redraw immediately the target object by taking into account the new value of the LineStyle attribute and according to its position in the *DisplayStack* of the active Application object.

Provisions of use:

- The *Target* object shall be an available LineArt object.
- *NewLineStyle* shall be set or refer to a positive integer value: 1, 2 or 3.

Syntax description:

```
SetLineStyle          -->   Target,
                            NewLineStyle
Target                -->   GenericObjectReference
NewLineStyle          -->   GenericInteger
```

SetLineColour (*NewColour*)

Change the colour of lines of a LineArt object.

Execute the following sequence of actions:

1. Set the value of the RefLineColour attribute to *NewColour*.
2. If the target object is active, redraw immediately the target object by taking into account the new value of the LineWidth attribute and according to its position in the *DisplayStack* of the active Application object.

NewColour may be either an absolute colour value or the zero-based index of a colour in the look-up table referenced by the *PaletteRef* attribute.

Provisions of use:

- The *Target* object shall be an available LineArt object.
- If RefLineColour is currently set to an absolute colour value, *NewColour* shall be set or refer to an absolute colour value.
- If RefLineColour is currently set to an index of a colour look-up table, *NewColour* shall be set or refer to an index of a colour look-up table.

Syntax description:

```
SetLineColour         -->   Target,
                            NewColour
Target                -->   GenericObjectReference
NewColour             -->   NewColourIndex | NewAbsoluteColour
NewColourIndex        -->   GenericInteger
NewAbsoluteColour     -->   GenericOctetString
```

SetFillColour (*NewColour*)

Change the fill-in colour of a LineArt object.

Execute the following sequence of actions:

1. Set the value of the RefFillColour attribute to *NewColour*.
2. If the target object is active, redraw immediately the target object by taking into account the new value of the RefFillColour attribute and according to its position in the *DisplayStack* of the active Application object.

NewColour may be either an absolute colour value or the zero-based index of a colour in the look-up table referenced by the *PaletteRef* attribute.

If *NewColour* is not encoded, the RefFillColour attribute shall be set to «transparent».

Provisions of use:

- The *Target* object shall be an available LineArt object.
- If *RefFillColour* is not currently set, *NewColour* shall be set or refer to an absolute colour value.
- If RefFillColour is currently set to an absolute colour value, *NewColour* shall be set or refer to an absolute colour value.
- If RefFillColour is currently set to an index of a colour look-up table, *NewColour* shall be set or refer to an index of a colour look-up table.

Syntax description:

```
SetFillColour          -->   Target,
                             NewColour?
Target                 -->   GenericObjectReference
NewColour              -->   NewColourIndex | NewAbsoluteColour
NewColourIndex         -->   GenericInteger
NewAbsoluteColour      -->   GenericOctetString
```

33.5 Formal description

```
LineArt Class              -->  Visible Class,
                                BorderedBoundingBox?,
                                OriginalLineWidth?,
                                OriginalLineStyle?,
                                OriginalRefLineColour?,
                                OriginalRefFillColour?
BorderedBoundingBox        -->  BOOLEAN
OriginalLineWidth          -->  INTEGER
OriginalLineStyle          -->  INTEGER
OriginalRefLineColour      -->  Colour
OriginalRefFillColour      -->  Colour
```

34 Rectangle Class

Description	Defines a data structure that deals with rectangles.
Base class	LineArt
Subclasses	None
Status	Concrete class

34.1 Attributes

This subclause defines the inherited, exchanged and internal attributes for this class.

34.1.1 Inherited attributes

This class has all the attributes of its base class, with the following constraints:

Attribute Name	Defined in	Constraints and Requirements
ContentHook	Ingredient	This attribute shall not be encoded for this class.
OriginalContent	Ingredient	This attribute shall not be encoded for this class.
BoxSize, Position	Visible	In addition to defining the bounding box for the Rectangle, these attributes also define the size of the Rectangle itself. The Rectangle's size shall be such that it precisely fits inside its bounding box. This means that the borders of the Rectangle occupy *LineWidth* pixels on the inside of the Rectangle's bounding box.
BorderedBoundingBox	LineArt	This attribute shall not be encoded for this class. It is always be interpreted as True. Unlike for its parent class, the BorderedBoundingBox of a Rectangle (i.e. the shape of the Rectangle) shall be drawn using LineWidth, LineStyle and RefLineColour, rather than OriginalLineWidth, OriginalLineStyle and OriginalRefLineColour.

34.1.2 Own exchanged attributes

This class defines no additional exchanged attributes.

34.1.3 Own internal attributes

This class defines no additional internal attributes.

34.2 Events

This class has the same events as its base class, with identical semantics.

34.3 Internal behaviours

The internal behaviours of this class are the same as those of its base class, with identical semantics.

34.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base class, with identical semantics.

34.5 Formal description

```
Rectangle Class          --> LineArt Class
```

35 DynamicLineArt Class

Description	Defines means to dynamically draw vectorial graphical objects
Base class	LineArt
Subclasses	None
Status	Concrete Class

35.1 Attributes

35.1.1 Inherited attributes

This class has all the attributes of its base class, with the following constraints:

Attribute Name	Defined in	Constraints and Requirements
ContentHook	Ingredient	This attribute shall not be encoded for this class.
OriginalContent	Ingredient	This attribute shall not be encoded for this class.

35.1.2 Own exchanged attributes

This class defines no additional exchanged attributes.

35.1.3 Own internal attributes

This class defines no additional internal attributes.

35.2 Events

This class has the same events as its base class, with identical semantics.

35.3 Internal Behaviours

This class defines no additional internal behaviour.

35.4 Effect of MHEG-5 Actions

This class has the same set of mheg-5 actions as its base class, with identical semantics. In addition, the following applicable MHEG-5 actions are defined:

SetBoxSize	This action redraws the DynamicLineArt object with its new size and cleared of all of its previous drawings, that is filled with *OriginalRefFillColour* (in the particular case where the new size is exactly the old size, this action is similar to a Clear action). When BorderedBoundingBox is set to True, the border is redrawn.
SetPosition, BringToFront, SendToBack, PutBefore, PutBehind	After each of these actions, the DynamicLineArt widget is and cleared of all of its previous drawings, that is filled with *OriginalRefFillColour.* When BorderedBoundingBox is set to True, the border is redrawn.

SetLineWidth — Setting a new value of *LineWidth* shall not modify line width of all existing graphics from the DynamicLineArt object, it shall be used to render the next graphics to be drawn.

SetLineStyle — Setting a new value of *LineStyle* shall not modify line style of all existing graphics from the DynamicLineArt object, it shall be used to render the next graphics to be drawn.

SetLineColour — Setting a new value of *RefLineColour* shall not modify the colour of all existing graphics from the DynamicLineArt object, it shall be used to render the next graphics to be drawn.

SetFillColour — Setting a new value of *RefFillColour* shall not modify colour of all existing graphics from the DynamicLineArt object, it shall be used to render the next graphics to be drawn.

GetLineWidth (*LineWidthVar*) — Returns the current *LineWidth* in the variable *LineWidthVar*.

Provisions of use:

- *LineWidthVar* shall refer to an active IntegerVariable object.
- The *Target* object shall be an available DynamicLineArt object.

Syntax description:

```
GetLineWidth          -->  Target,
                           LineWidthVar
Target                -->  GenericObjectReference
LineWidthVar          -->  ObjectReference
```

GetLineStyle (*LineStyleVar*) — Returns the current *LineStyle* into the variable *LineStyleVar*.

Provisions of use:

- *LineStyleVar* shall refer to an active IntegerVariable object.
- The *Target* object shall be an available DynamicLineArt object.

Syntax description:

```
GetLineStyle          -->  Target,
                           LineStyleVar
Target                -->  GenericObjectReference
LineStyleVar          -->  ObjectReference
```

GetLineColour (*LineColourVar*) — Returns the current *LineColour* in the variable *LineColourVar*

Provisions of use:

- *LineColourVar* shall refer to an active OctetStringVariable object if the RefLineColour is specified as an OctetString. *LineColourVar* shall refer to an

active IntegerVariable object if the RefLineColour is specified as an Integer.

- The *Target* object shall be an available DynamicLineArt object.

Syntax description:

```
GetLineColour          -->  Target,
                            LineColourVar
Target                 -->  GenericObjectReference
LineColourVar          -->  ObjectReference
```

GetFillColour (*FillColourVar*)

Returns the current *RefFillColour* in the variable *FillColourVar*

Provisions of use:

- *FillColourVar* shall refer to an active OctetStringVariable object if the RefFillColour is specified as an OctetString. *FillColourVar* shall refer to an active IntegerVariable object if the RefFillColour is specified as an Integer..
- The *Target* object shall be an available DynamicLineArt object.

Syntax description:

```
GetFillColour          -->  Target,
                            FillColourVar
Target                 -->  GenericObjectReference
FillColourVar          -->  ObjectReference
```

DrawArc (*X, Y, EllipseWidth, EllipseHeight, StartAngle, ArcAngle*)

Draws an arc between *StartAngle* and *StartAngle + ArcAngle*. (arc BC in Figure 12 - Illustration of drawing parameters below).

Arc is drawn in *LineColour*.

Point *X, Y* is relative to the *Position* attribute of the object. That is to say that X=0, Y=0 corresponds to the top left corner of the bounding box. Values of X and Y outside of the bounding box are allowed, but only the part of the drawing within the bounding box shall be rendered.

Angles are expressed in 64ths of degrees and are in the interval [0, 23039]. *ArcAngle* shall not be 0.

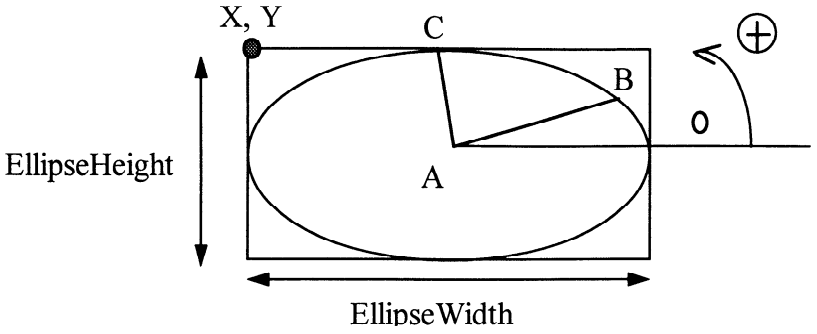


Figure 12 - Illustration of drawing parameters

Provisions of use:

- The *Target* object shall be an available DynamicLineArt object.

Syntax description:

```
DrawArc                 -->  Target,
                             X,
                             Y,
                             EllipseWidth,
                             EllipseHeight,
                             StartAngle,
                             ArcAngle
Target                  -->  GenericObjectReference
X                       -->  GenericInteger
Y                       -->  GenericInteger
EllipseWidth            -->  GenericInteger
EllipseHeight           -->  GenericInteger
StartAngle              -->  GenericInteger
ArcAngle                -->  GenericInteger
```

DrawSector (*X, Y, EllipseWidth, EllipseHeight, StartAngle, ArcAngle*)

Draws a sector between *StartAngle* and *StartAngle + ArcAngle*. (the surface ABC in Figure 12 - Illustration of drawing parameters above).

Lines are drawn with *RefLineColour* and the surface is filled up with *RefFillColour*.

Point *X, Y* is relative to the *Position* attribute of the object. That is to say that X=0, Y=0 corresponds to the top left corner of the bounding box. Values of X and Y outside of the bounding box are allowed, but only the part of the drawing within the bounding box shall be rendered.

Angles are expressed in 64ths of degrees and are in the interval [0, 23039]. *ArcAngle* shall not be 0.

Provisions of use:

- The *Target* object shall be an available DynamicLineArt object.

Syntax description:

```
DrawSector              -->  Target,
                             X,
                             Y,
                             EllipseWidth,
                             EllipseHeight,
                             StartAngle,
                             ArcAngle
Target                  -->  GenericObjectReference
X                       -->  GenericInteger
Y                       -->  GenericInteger
EllipseWidth            -->  GenericInteger
EllipseHeight           -->  GenericInteger
```

```
StartAngle                  -->  GenericInteger
ArcAngle                    -->  GenericInteger
```

DrawLine
(X1, Y1, X2, Y2)

Draws a line between (*X1, Y1*) and (*X2, Y2*)

Points *X1, Y1 and X2, Y2* are relative to the *Position* attribute of the object. Values outside of the bounding box are allowed, but only the part of the drawing within the bounding box shall be rendered.

Provisions of use:

- The *Target* object shall be an available DynamicLineArt object.

Syntax description:

```
DrawLine                    -->  Target,
                                 X1,
                                 Y1,
                                 X2,
                                 Y2
Target                      -->  GenericObjectReference
X1                          -->  GenericInteger
Y1                          -->  GenericInteger
X2                          -->  GenericInteger
Y2                          -->  GenericInteger
```

DrawOval
(X, Y, EllipseWidth, EllipseHeight)

Draws an ellipse bounded by the rectangle defined by the parameters (see Figure 12 - Illustration of drawing parameters above).

This ellipse is filled up with *RefFillColour*.

Point *X, Y* is relative to the *Position* attribute of the object. That is to say that X=0, Y=0 corresponds to the top left corner of the bounding box. Values of X and Y outside of the bounding box are allowed, but only the part of the drawing within the bounding box shall be rendered.

Provisions of use:

- The *Target* object shall be an available DynamicLineArt object.

Syntax description:

```
DrawOval                    -->  Target,
                                 X,
                                 Y,
                                 EllipseWidth,
                                 EllipseHeight
Target                      -->  GenericObjectReference
X                           -->  GenericInteger
Y                           -->  GenericInteger
EllipseWidth                -->  GenericInteger
```

```
EllipseHeight                --> GenericInteger
```

DrawPolygon (*PointList*)

Draw a closed polygon.

This polygon is filled up with *RefFillColour*.

PointList is a list of Point.

A Point is defined by X, Y coordinate, relative to the *Position* attribute of the object. Values of X and Y outside of the bounding box are allowed, but only the part of the drawing within the bounding box shall be rendered.

Provisions of use:

- The *Target* object shall be an available DynamicLineArt object.

Syntax description:

```
DrawPolygon                  --> Target,
                                 PointList
Target                       --> GenericObjectReference
PointList                    --> Point+
Point                        --> X,
                                 Y
X                            --> GenericInteger
Y                            --> GenericInteger
```

DrawPolyline (*PointList*)

Draws series of joint lines.

PointList is a list of Point.

A Point is defined by X, Y coordinate, relative to the *Position* attribute of the object. Values of X and Y outside of the bounding box are allowed, but only the part of the drawing within the bounding box shall be rendered.

Provisions of use:

- The *Target* object shall be an available DynamicLineArt object.

Syntax description:

```
DrawPolyline                 --> Target,
                                 PointList
Target                       --> GenericObjectReference
PointList                    --> Point+
Point                        --> X,
                                 Y
X                            --> GenericInteger
Y                            --> GenericInteger
```

DrawRectangle *(X1, Y1, X2, Y2)*

Draws a rectangle.

This rectangle is filled up with *RefFillColour*.

Top Left point is (*X1, Y1*) and Bottom Right point is (*X2, Y2*). (*X1, Y1*) and (*X2, Y2*) are relative to the *Position* attribute of the object. Values outside of the bounding box are allowed, but only the part of the drawing within the bounding box shall be rendered.

Provisions of use:

- The *Target* object shall be an available DynamicLineArt object.

Syntax description:

```
DrawRectangle          -->  Target,
                            X1,
                            Y1,
                            X2,
                            Y2
X1                     -->  GenericInteger
Y1                     -->  GenericInteger
X2                     -->  GenericInteger
Y2                     -->  GenericInteger
```

Clear Fills up the bounding box with the *OriginalRefFillColour*.

When BorderedBoundingBox is set to True, the border is not filled with *OriginalRefFillColour*.

Provisions of use:

- The *Target* object shall be an available DynamicLineArt object.

Syntax description:

```
Clear                  -->  Target
Target                 -->  GenericObjectReference
```

35.5 Formal description

```
DynamicLineArt Class    -->  LineArt Class
```

36 Text Class

Description	Defines attributes and behaviour of pieces of textual information.
Base class	Visible
Subclasses	EntryField, HyperText
Status	Concrete class

36.1 Attributes

This subclause defines the inherited, exchanged and internal attributes for this class.

36.1.1 Inherited attributes

This class has all the attributes of its base class, with the following constraints:

Attribute Name	Defined in	Constraints and Requirements
ContentHook	Ingredient	Either this attribute or its corresponding default attribute in the Application class (TextContentHook) is mandatory for this class.
OriginalContent	Ingredient	This attribute is mandatory for this class.

36.1.2 Own exchanged attributes

This class defines the following additional exchanged attributes:

OriginalFont

Indication of which font to use when initially presenting the Text object.

The OriginalFont attribute represents either a name for a font (which is resident in the MHEG-5 engine) or a reference to a Font object. In both cases, the indicated font shall be used for rendering the Text object.

When no font reference is encoded, the Text object shall be presented using the default font referenced in the active Application object, if no font is referenced there, a default font of the MHEG-5 engine shall be used.

NOTE - The OriginalFont attribute provides the initial font for a Text object. In addition, the text encoding format may contain escape sequences to switch between fonts.

- Optional attribute.
- OctetString representing a FontName, or reference to a Font object.
- Default value: Value encoded by Application.

FontAttributes

This attribute is used to set specific Font attributes such as style, character size, text colour and background colour.

The exact encoding format of the *FontAttributes* attribute is related to the value of the type of Font object mentioned by the *Font* attribute.

When no FontAttributes is encoded, the Text object shall be presented using the default *FontAttributes* encoded in the active Application object, if no *FontAttributes* is referenced there, no specific attributes are set.

- Optional OctetString.
- Default value: Value encoded by Application.

TextColour

Indicate which colour should be used to render the foreground of the text object. This attribute is interpreted as a zero-based index in the colour look-up table defined by the *PaletteRef* attribute, or as a direct colour value, depending on the attribute type.

- Optional Integer or OctetString. An Integer will be interpreted as an index in a Palette; an OctetString will be interpreted as a direct colour value.
- Default value: Value encoded by Application.

BackgroundColour

Indicate which colour should be used to render the background of the text object. This attribute is interpreted as a zero-based index in the colour look-up table defined by the *PaletteRef* attribute, or as a direct colour value, depending on the attribute type.

- Optional Integer or OctetString. An Integer will be interpreted as an index in a Palette; an OctetString will be interpreted as a direct colour value.
- Default value: Value encoded by Application.

CharacterSet

Identification of the character set, or set of character sets, that shall be used by default for Text rendering. This Integer shall be encoded with a value representing the character set. The application domain shall define a range of CharacterSet and its semantics.

NOTE - The *CharacterSet* attribute provides the initial character set for a Text object. In addition, the text encoding format may contain escape sequences to switch between character sets.

- Optional Integer.
- Default value: The value of *CharacterSet* attribute from the Application object, if that attribute is specified.

Horizontal-Justification

The *HorizontalJustification* attribute indicates how the text lines are justified relative to the vertical edges of the bounding box defined by the *BoxSize* and *Position* attributes of the Text object.

This attribute may be ignored if a coded representation of the text itself has the same functionality to specify this type of rendering. The application domain based on this part of ISO/IEC 13522 shall define each ContentHook for which the attribute be ignored.

- Optional attribute - one of *start* / *end* / *centre* / *justified.*
- Default value: *start.*

VerticalJustification

The *VerticalJustification* attribute indicates how the text lines are justified relatively to the horizontal edges of the bounding box defined by the *BoxSize* and *Position* attributes of the Text object.

This attribute may be ignored if a coded representation of the text itself has the same functionality to specify this type of rendering. The application domain based on this part of ISO/IEC 13522 shall define each ContentHook for which the attribute be ignored.

- Optional attribute - one of *start* / *end* / *centre* / *justified.*
- Default value: *start.*

LineOrientation

The *LineOrientation* attribute is combined with the *StartCorner* attribute to determine

the way characters are organised into lines, and lines into sequences of lines.

This attribute may be ignored if a coded representation of the text itself has the same functionality to specify this type of rendering. The application domain based on this part of ISO/IEC 13522 shall define each ContentHook for which the attribute be ignored.

- Optional attribute - one of *vertical* | *horizontal*.
- Default value: *horizontal*.

NOTE - For *vertical* line orientation, the character orientation is supposed to be normal (i.e., the entire line is rotated 90 degrees from *horizontal)*, unless the contents of the *Text* specify otherwise.

StartCorner

The *StartCorner* attribute contains an identification of the corner of the presentation area where the text rendering should start.

This attribute may be ignored if a coded representation of the text itself has the same functionality to specify this type of rendering. The application domain based on this part of ISO/IEC 13522 shall define each ContentHook for which the attribute be ignored.

- Optional attribute - one of *upper-left* | *upper-right* | *lower-left* | *lower-right*.
- Default value: *upper-left*.

TextWrapping

Indicate whether the text is wrapped at the end of the line, or whether it is clipped.

If the TextWrapping attribute is set to True, the text is wrapped. If it is set to False, it is clipped.

This attribute may be ignored if a coded representation of the text itself has the same functionality to specify this type of rendering. The application domain based on this part of ISO/IEC 13522 shall define each ContentHook for which the attribute be ignored.

- Optional Boolean.
- Default value: False.

36.1.3 Own internal attributes

This class defines the following additional internal attributes:

TextData

Value of the textual content data of the Text object.

If the *Content* attribute of the Text object is an IncludedContent, *TextData* is initially set to IncludedContent.

If the *Content* attribute of the Text object is a reference to an external data source, *TextData* is initially set to an OctetString representing the content of this external data source. In this case, the value of the ContentHook attribute might be used to format the value of *TextData*.

- OctetString.
- Initial value: IncludedContent or the content of a ReferencedContent.

Font

Font to use when presenting the Text object.

The *Font* attribute represents either a name for a font (which is resident in the MHEG-5 engine) or a reference to a Font object. In both cases, the indicated font shall be used for rendering the Text object.

NOTE - The Font attribute provides the initial font for a Text object. In addition, the text encoding format may contain escape sequences to switch between fonts.

- Optional attribute.

- OctetString representing a FontName, or reference to a Font object.
- Initial value: Value of the OriginalFont attribute.

36.2 Events

This class has the same events as its base class, with identical semantics.

36.3 Internal behaviours

The internal behaviours of this class have the same semantics as for its base class.

36.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base class, with identical semantics, except for SetData. In addition, the following applicable MHEG-5 actions are defined:

SetData — Execute synchronously the following actions:

1. Update the value of the *TextData* internal attribute of the target Text.
2. Execute the SetData action as defined in the Presentable class.

Provisions of use and syntax description are provided in the Ingredient class.

GetTextContent (*TextContentVar*) — Set the Variable referenced by *TextContentVar* to the value of the *Content* attribute.

NOTE - If the *Content* attribute of the Text object is included, *TextContentVar* returns the text OctetString; if the *Content* attribute of the Text object is referenced, *TextContentVar* returns that object reference.

Provisions of use:

- *TextContentVar* shall refer to an active OctetStringVariable object or an active ContentRefVariable object.
- The *Target* object shall be an available Text object.

Syntax description:

```
GetTextContent          -->  Target,
                             TextContentVar
Target                  -->  GenericObjectReference
TextContentVar          -->  ObjectReference
```

GetTextData (*TextDataVar*) — Set the Variable referenced by *TextDataVar* to the value of the *TextData* attribute.

Provisions of use:

- *TextDataVar* shall refer to an active OctetString object.
- The *Target* object shall be an available Text object.

Syntax description:

```
GetTextData             -->  Target,
```

```
                                TextDataVar
Target                  -->     GenericObjectReference
TextDataVar             -->     ObjectReference
```

SetFontRef (*NewFont*)

Change the character font used to present a text.

Execute the following sequence of actions:

1. Set the value of the *Font* attribute to *NewFont*.
2. If the target Text object is active, redraw the target object immediately by taking into account the new value of *Font* and according to its position in the *DisplayStack* of the active Application object.

NewFont might be a reference to a Font object or a font name. This part of ISO/IEC 13522 does not define how font names are managed by the MHEG-5 engine.

Provisions of use:

- The *Target* object shall be an available Text object.
- If *NewFont* references a Font object, this Font object shall be available.

Syntax description:

```
SetFontRef              -->     Target,
                                NewFont
Target                  -->     GenericObjectReference
NewFont                 -->     NewFontName | NewFontReference
NewFontName             -->     GenericOctetString
NewFontReference        -->     GenericObjectReference
```

36.5 Formal description

```
Text Class              -->     Visible Class,
                                OriginalFont?,
                                FontAttributes?,
                                TextColour?,
                                BackgroundColour?,
                                CharacterSet?,
                                HorizontalJustification?,
                                VerticalJustification?,
                                LineOrientation?,
                                StartCorner?,
                                TextWrapping?
OriginalFont            -->     OctetString | ObjectReference
FontAttributes          -->     OctetString
TextColour              -->     Colour
BackgroundColour        -->     Colour
CharacterSet            -->     INTEGER
Horizontal-             -->     start | end | centre | justified
Justification
VerticalJustification   -->     start | end | centre | justified
LineOrientation         -->     vertical | horizontal
```

StartCorner	-->	*upper-left	upper-right	lower-left	lower-right*
TextWrapping	-->	*BOOLEAN*

37 Stream Class

Description	Defines the behaviour of a composition of continuous media (Video, Audio and RTGraphics) that are presented in synchronisation.
Base class	Presentable
Subclasses	None
Status	Concrete class

37.1 Attributes

This subclause defines the inherited, exchanged and internal attributes for this class.

37.1.1 Inherited attributes

This class has all the attributes of its base class, with the following constraints:

Attribute Name	Defined in	Constraints and Requirements
ContentHook	Ingredient	Either this attribute or its corresponding default attribute in the Application class (StreamContentHook) is mandatory for this class.
OriginalContent	Ingredient	This attribute is mandatory for this class. It contains a reference to a whole multiplex of synchronised media.

37.1.2 Own exchanged attributes

This class defines the following additional exchanged attributes:

Multiplex

List of inclusion of Video, Audio and RTGraphics objects that are intended to be presented simultaneously. These are called StreamComponents below. Each stream component has a tag, which is used to identify it uniquely within the stream.

The *InitiallyActive* attribute of each StreamComponent determines whether the corresponding elementary stream is automatically played as a result of the Stream multiplex being activated for the first time.

Note that all behaviour associated with synchronisation is always accessed via the Stream class; for example, it is not possible to target the SetCounterPosition action to an Audio object.

- Sequence of inclusions of Video, Audio and RT-Graphics objects

Storage

Indicate whether the composition of continuous media is loaded into memory before rendering or if it is presented directly off the stream coming, for instance, from a server. For the MHEG-5 engine, the difference in handling is that in the *memory* case, the MHEG-5 engine shall synchronise the stream, whereas in the *stream* case, the stream is synchronised by the server.

- Optional attribute - one of *memory* / *stream.*
- Default value: *stream.*

Looping

Number of performances of the Stream object.

In the counting of loops, actual counting takes place as the Stream reaches its end. As a result SetCounterPosition action shall be interpreted in a given loop.

When a Stream is played and stopped before all the loops are done, the next play action shall continue from that loop (and from that CounterPosition, unless it is set by a specific action); in other words the attribute Looping shall represent the total number of loops.

When a Stream has reached the end of all loops and is stopped, a new activation shall play the Stream as if it were activated for the first time, that is looping according to the Looping attribute.

- Optional Integer.
- Default value: 1.
- Special value: 0 means *infinity*.

37.1.3 Own internal attributes

This class defines the following additional internal attributes:

Speed Rate at which the composition of continuous media is presented.

The *Speed* attribute is a rational number, represented as two Integers *(a, b)*. Speed is defined as *a/b*. The semantics of the following values differ depending on whether the MHEG-5 engine has access to an underlying layer to support trick modes.

NOTE - Trick modes might be provided by the ISO/IEC 13818-6 DSM-CC protocol.

	No Trick Mode	*Trick Mode*
-1/1	Treated as 0.	Reverse. Play backwards at normal speed.
0/1	Stop. Freeze at current position.	Stop. Freeze at current position.
1/1	Normal play. Play from a point that is not under the control of the MHEG-5 engine.	Normal play. Play from the point where the stream was when it left its previous mode.

Other values of Speed are allowed (e.g., 1/2 or 2/1) in trick mode. In the case of no trick mode, such other value of Speed shall be treated as normal play (1/1) when they are positive and as stop (0/1) when they are negative or null.

- Pair of Integers: a Numerator and an optional Denominator that defaults to 1.
- Default value: 1/1.

CounterPosition Current temporal position of the Stream within the duration of a stream at normal speed.

This attribute is expressed in StreamCounterUnits.

The actual definition of the StreamCounterUnit is out of the scope of this part of ISO/IEC 13522.

- Integer.
- Default value: 0.

CounterEndPosition Position of the last frame of a stream played at normal speed. The stream will stop automatically when it hits this position. The *StreamStopped* event is generated.

This attribute is expressed in StreamCounterUnits.

The actual definition of the StreamCounterUnit is out of the scope of this part of ISO/IEC 13522.

- Integer.
- Default value: -1, meaning EndOfStream.

CounterTriggers List of values representing the counter positions of the Stream where the stream player shall generate *CounterTrigger* events for this Stream.

Each trigger has a unique identifier within the *CounterTriggers* list and a counter position expressed in StreamCounterUnits.

- Sequence of the following data structures:
  - trigger identifier: Integer,
  - counter position: Integer.
- Initial value: Empty sequence.

37.2 Events

This class has the same events as its base class, with identical semantics. In addition, the following events are defined:

StreamEvent This event is automatically generated by the Stream player when the Stream multiplex crosses a specific marker. The marker is recognised on the basis of a tag that may be encoded within the Stream content data structure. There might be several markers with the same identity along a Stream multiplex.

- Associated data: StreamEventTag - OctetString.

NOTES

1 According to the semantics of the Link class, a Link set to trigger on StreamEvents from a specific stream object that does not have its EventData set will trigger on all StreamEvents from that object.

2 The encoding of StreamEvents may be done using the ISO/IEC 13818-6 DSM-CC protocol.

StreamPlaying This event is generated when a Stream multiplex has started playing. More specifically, it is generated simultaneously with the first piece of content data (video frame, audio sample) being presented to the user.

- No associated data.

StreamStopped This event is generated when a Stream multiplex has stopped playing. More specifically, it is generated as soon as the last piece of content data (video frame, audio sample) has been presented to the user. Note that the *RunningStatus* of the Stream object is not affected by the occurrence of a *StreamStopped* event.

- No associated data.

CounterTrigger This event is automatically generated by the Stream player when the *CounterPosition* of the Stream object crosses a value set with the SetCounterTrigger action. There might be several CounterTriggers triggered at the same counter position of a Stream.

- Associated data: TriggerIdentifier - Integer.

NOTE - The encoding of counter position within the stream may be done using the

ISO/IEC 13818-6 DSM-CC protocol.

37.3 Internal behaviours

The following internal behaviours semantics have changed from this object's base class:

Preparation Execute the following sequence of actions:

1. Apply the first three steps of the *Preparation* behaviour as defined in the Root class.
2. Apply the *Activation* behaviour to all StreamComponents of the Stream that have the InitiallyActive attribute set to True, in the order they appear in the Stream Multiplex.
3. Apply steps four to six of the *Preparation* behaviour as defined in the Root class.

Destruction Execute the following sequence of actions:

1. Apply the *Destruction* behaviour to all StreamComponents of the Stream, in the reverse order they appear in the Stream Multiplex.
2. Apply the *Destruction* behaviour as defined in the base class.

Activation Execute the following sequence of actions:

1. Apply the *Activation* behaviour as defined in the base class.
2. Start playing all active StreamComponents.
3. Set the *RunningStatus* attribute to True and generate an *IsRunning* event.

The Activation and Deactivation of StreamComponents like Audio, Video or RTGraphics corresponds to enabling or disabling them in a stream. For instance when a Stream plays, activating an Audio shall make that component audible (in addition to the others).

Deactivation Execute the following sequence of actions:

1. If the RunningStatus attribute of the object is False, ignore the behaviour, otherwise:
2. Stop playing all active StreamComponents.
3. Execute *Deactivation* behaviour as defined in the base class.

37.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base class, with identical semantics. In addition, the following applicable MHEG-5 actions are defined:

SetData, Clone — SetData and Clone shall not be targeted to Stream.

SetCounterTrigger (*TriggerIdentifier*, *NewCounterValue*) — Update the list of CounterTriggers of a Stream object.

Execute the following sequence of actions:

1. Update the set of triggers of the *CounterTriggers* list of the target Stream, according to the following rules:

a) If *TriggerIdentifier* is the identifier of an existing trigger in *CounterTriggers*, the new trigger replaces the previous one.

b) If there is no trigger with identifier *TriggerIdentifier* in *CounterTriggers*, insert a new trigger with identifier *TriggerIdentifier* and value *NewCounterValue* in *CounterTriggers*.

c) If *NewCounterValue* is not encoded and there is a trigger with identifier *TriggerIdentifier* in *CounterTriggers*, remove this trigger from the *CounterTriggers* list.

d) If *NewCounterValue* is not encoded and there is no trigger with identifier *TriggerIdentifier* in *CounterTriggers*, discard this action.

2. If the target Stream object is active, the MHEG-5 engine shall generate *CounterTrigger* events according to the new value of the *CounterTriggers* list.

Provisions of use:

- The *Target* object shall be an available Stream object.

Syntax description:

```
SetCounterTrigger     -->  Target,
                           TriggerIdentifier,
                           NewCounterValue?
Target                -->  GenericObjectReference
TriggerIdentifier     -->  GenericInteger
NewCounterValue       -->  GenericInteger
```

SetSpeed (*NewSpeed*)

Change the presentation speed of a stream.

Execute the following sequence of actions:

1. Set the value of the *Speed* attribute of the target Stream object to *NewSpeed*.
2. If the target Stream is active, update immediately the rendering of the stream by taking into account the new value of the *Speed* attribute.

The *NewSpeed* attribute is defined as a ratio Numerator/Denominator.

NOTES:

1 As mentioned earlier, in a broadcast environment, play and stop shall start when possible in the broadcasted stream. In other cases, setting new speed (including normal play and stop) shall take place at the current counter position, or as close to it as possible (e.g. next I-frame). Counter position can be set by the appropriate action.

2 If *Trick Modes* are not supported by the engine, the value of *Speed* can still be set to any value, the engine will interpret them as explained in the *Speed* internal attribute.

Provisions of use:

- The *Target* object shall be an available Stream object.

Syntax description:

```
SetSpeed              -->  Target,
```

```
                                NewSpeed
Target                  -->     GenericObjectReference
NewSpeed                -->     Rational
Rational                -->     Numerator,
                                Denominator?
Numerator               -->     GenericInteger
Denominator             -->     GenericInteger
```

SetCounterPosition (*NewCounter-Position*)

Change the current position within a stream.

Execute the following sequence of actions:

1. If the MHEG-5 engine is not provided with an underlying presentation layer that supports trick modes, discard this action.
2. Set the value of the *CounterPosition* attribute of the target Stream to *NewCounterPosition*.
3. If the target Stream is active, skip immediately to the new position without changing the *RunningStatus* of the target Stream.

NOTE - Stream events are not generated because of a SetCounterPosition action.

Provisions of use:

- The *Target* object shall be an available Stream object.
- *NewCounterPosition* shall indicate a valid position within the target Stream.

Syntax description:

```
SetCounterPosition      -->     Target,
                                NewCounterPosition
Target                  -->     GenericObjectReference
NewCounterPosition      -->     GenericInteger
```

SetCounterEnd-Position (*NewCounterEnd-Position*)

Change the end position of a stream.

Execute the following sequence of actions:

1. If the MHEG-5 engine is not provided with an underlying presentation layer that supports trick modes, discard this action.
2. Set the value of the *CounterEndPosition* attribute of the target Stream to *NewCounterEndPosition*.
3. If the target Stream is active and *NewCounterEndPosition* is already passed, stop the target Stream.

Provisions of use:

- The *Target* object shall be an available Stream object.
- *NewCounterPosition* shall indicate a valid position within the target Stream.

Syntax description:

```
SetCounterPosition      -->     Target,
                                NewCounterPosition
```

```
Target                  -->  GenericObjectReference
NewCounterPosition      -->  GenericInteger
```

37.5 Formal description

```
Stream Class            -->  Presentable Class,
                             Multiplex,
                             Storage?,
                             Looping?
Multiplex               -->  StreamComponent+
StreamComponent         -->  Audio class | Video class |
                             RTGraphics class
Storage                 -->  memory | stream
Looping                 -->  INTEGER
```

38 Audio Class

Description	Defines the attributes and behaviour of an elementary audio stream of a Stream multiplex. The Audio object shall be a StreamComponent of a Stream object.
Base class	Presentable
Subclasses	None
Status	Concrete class

38.1 Attributes

This subclause defines the inherited, exchanged and internal attributes for this class.

38.1.1 Inherited attributes

This class has all the attributes of its base class, with the following constraints:

Attribute Name	Defined in	Constraints and Requirements
RunningStatus	Root	This attribute expresses that this object is enabled or disabled to play when the Stream object containing it is active. The object is only played when this attribute is True and the RunningStatus of the Stream is True.
ContentHook, OriginalContent, Shared	Ingredient	These attributes shall not be encoded for this class.

38.1.2 Own exchanged attributes

This class defines the following additional exchanged attribute:

OriginalVolume — Volume of the Audio object when it is first available.

The OriginalVolume attribute is expressed in dB, where 0 dB is defined to be the standard volume for playing back audio. The precise accuracy of the volume rendering is outside the scope of this part of ISO/IEC 13522.

- Optional Integer.
- Default value: 0.

ComponentTag — A unique identifier for the elementary audio stream within a stream of multiplexed media.

- Integer.

38.1.3 Own internal attributes

This class defines the following additional internal attribute:

Volume — Current volume of the Audio object, defined in the same way as OriginalVolume.

- Integer.
- Initial value: *OriginalVolume.*

38.2 Events

This class has the same events as its base class, with identical semantics.

38.3 Internal behaviours

This class defines no additional internal behaviours.

38.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base class, with identical semantics. In addition, the following applicable MHEG-5 actions are defined:

SetData, Clone

SetData and Clone shall not be targeted to Audio.

SetVolume (*NewVolume*)

Change the volume of an audio.

Execute the following sequence of actions:

1. Set the *Volume* attribute of the target Audio object to *NewVolume*.
2. If the elementary audio stream identified by the ComponentTag attribute is being played, update the rendering of the Audio by taking into account the new *Volume*.

Provisions of use:

- The *Target* object shall be an available Audio object.

Syntax description:

```
SetVolume          -->   Target,
                         NewVolume
Target             -->   GenericObjectReference
NewVolume          -->   GenericInteger
```

GetVolume (*VolumeVar*)

Returns the volume of an Audio.

Provisions of use:

- The *Target* object shall be an available Audio object.
- *VolumeVar* shall be an active IntegerVariable object.

Syntax description:

```
GetVolume          -->   Target,
                         VolumeVar
Target             -->   GenericObjectReference
VolumeVar          -->   ObjectReference
```

38.5 Formal description

```
Audio Class              -->  Presentable Class,
                              ComponentTag
                              OriginalVolume?
ComponentTag             -->  INTEGER
OriginalVolume           -->  INTEGER
```

39 Video Class

Description	Defines the attributes and behaviour of an elementary video stream of a Stream multiplex. The Video object shall be a StreamComponent of a Stream object.
Base class	Visible
Subclasses	None
Status	Concrete class

39.1 Attributes

This subclause defines the inherited, exchanged and internal attributes for this class.

39.1.1 Inherited attributes

This class has all the attributes of its base class, with the following constraints:

Attribute Name	Defined in	Constraints and Requirements
RunningStatus	Root	This attribute expresses that this object is enabled or disabled to play when the Stream object containing it is active. The object is only played when this attribute is True and the RunningStatus of the Stream is True.
ContentHook, OriginalContent, Shared	Ingredient	These attributes shall not be encoded for this class.
OriginalPaletteRef	Visible	This attribute shall not be encoded for this class.

39.1.2 Own exchanged attributes

This class defines the following additional exchanged attributes:

Termination — This attribute indicates whether the last frame of the video shall disappear when the presentation of the video finishes, or whether it shall freeze.

- Optional attribute - One of *freeze* | *disappear*.
- Default value: *disappear*.

ComponentTag — A unique identifier for the elementary video stream within a stream of multiplexed media.

- Integer.

39.1.3 Own internal attributes

This class defines no additional internal attribute.

39.2 Events

This class has the same events as its base class, with identical semantics.

39.3 Internal behaviours

This class defines no additional internal behaviours.

39.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base class, with identical semantics. In addition, the following applicable MHEG-5 actions are defined:

SetData, Clone — SetData and Clone shall not be targeted to Video.

ScaleVideo (*XScale*, *YScale*) — If the MHEG-5 engine implements the Scaling option, the effect of this action is to adapt the rendering of the Video so that it fits to the *XScale* and *YScale* dimensions.

XScale and YScale parameters represent the final dimensions of the Video in pixel numbers. Thus, the graphical representation of the Video may not keep its original aspect ratio.

Note that this action does not affect the *BoxSize* internal attribute of the Video object.

Provisions of use:

- The *Target* object shall be an available Video object.
- *XScale* and *YScale* shall be positive Integers.

Syntax description:

```
ScaleVideo          -->  Target,
                         XScale, YScale
Target              -->  GenericObjectReference
XScale, YScale      -->  GenericInteger
```

39.5 Formal description

```
Video Class             -->  Visible Class,
                             ComponentTag
                             Termination?
Termination             -->  freeze | disappear
ComponentTag            -->  INTEGER
```

40 RTGraphics Class

Description	Defines the attributes and behaviour of non-persistent graphics objects, defined as of an elementary stream of a Stream multiplex. The RTGraphics object shall be a StreamComponent of a Stream object.
Base class	Visible
Subclasses	None
Status	Concrete class

40.1 Attributes

This subclause defines the inherited, exchanged and internal attributes for this class.

40.1.1 Inherited attributes

This class has all the attributes of its base class, with the following constraints:

Attribute Name	Defined in	Constraints and Requirements
RunningStatus	Root	This attribute expresses that this object is enabled or disabled to play when the Stream object containing it is active. The object is only played when this attribute is True and the RunningStatus of the Stream is True.
ContentHook, OriginalContent, Shared	Ingredient	These attributes shall not be encoded for this class.

40.1.2 Own exchanged attributes

This class defines the following additional exchanged attributes:

ComponentTag — A unique identifier for the elementary RTGraphics stream within a stream of multiplexed media.

- Integer.

Termination — This attribute indicates whether the last image of the graphics shall disappear when the presentation finishes, or whether it shall freeze.

- Optional attribute - One of *freeze* | *disappear*.
- Default value: *disappear.*

40.1.3 Own internal attributes

This class defines no additional internal attribute.

40.2 Events

This class has the same events as its base class, with identical semantics.

40.3 Internal behaviours

This class defines no additional internal behaviours.

40.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base class, with identical semantics. In addition, the following applicable MHEG-5 actions are defined:

SetData, Clone	SetData and Clone shall not be targeted to RTGraphics.

40.5 Formal description

```
RTGraphics Class        -->  Visible Class,
                             ComponentTag
                             Termination?
ComponentTag            -->  INTEGER
Termination             -->  freeze | disappear
```

41 Interactible Class

Description	Defines functionality associated with an interaction behaviour of Visibles.
Base class	None (mix-in class)
Subclasses	Slider, HyperText, EntryField, Button
Status	Abstract class

41.1 Attributes

This subclause defines inherited, exchanged and internal attributes for this class.

41.1.1 Inherited attributes

This class has no inherited attributes.

41.1.2 Own exchanged attributes

This class defines the following exchanged attributes:

EngineResp

Determine where the responsibility lies for generating visual feedback to the user as the result of changes to the internal attribute *HighlightStatus*.

If *EngineResp* is set to True, the MHEG-5 engine shall generate visual feedback to the user when there is a state change in the internal attribute *HighlightStatus*. When *EngineResp* is set to False, the engine shall generate no such visual feedback.

The exact nature of the visual feedback is outside the scope of this part of ISO/IEC 13522.

- Optional Boolean.
- Default value: True.

HighlightRefColour

Reference colour for visual feedback generated when the *HighlightStatus* attribute is True. The actual colour used for rendering the highlight is outside the scope of this part of ISO/IEC 13522; however, it is recommended that an MHEG-5 engine attempts to render the highlight using *HighlightRefColour.*

If the OriginalPaletteRef attribute is encoded, the HighlightRefColour attribute shall be a zero-based index of a colour in the colour look-up table defined by that Palette. Otherwise, it shall be an OctetString encoding an actual colour.

- Optional Integer or OctetString.
- Default value: Value encoded by Application.

41.1.3 Own internal attributes

This class defines the following internal attributes:

HighlightStatus

This attribute is associated with a certain type of visual feedback to the user.

When both *HighlightStatus* and *EngineResp* are True, the MHEG-5 engine shall generate a visual feedback to the user, for instance in the form of a line drawn around the Interactible object. In all other cases, no such feedback shall be generated. For the generation of this visual feedback, the HighlightRefColour may be

used.

NOTES

1 Although this visual feedback itself does not change the behaviour of the object, this attribute may be used to signal to the user that the Interactible is ready to be interacted with, for instance in an implementation of jumping-highlight navigation.

2 The only way to change the *HighlightStatus* attribute is through the SetHighlightStatus action.

- Boolean value.
- Initial value: False.

InteractionStatus This attribute describes whether or not the Interactible is currently being interacted with.

If *InteractionStatus* is False, the Interactible is not currently being interacted with by the user. Handling of user input events takes place normally. See clause 53.

If *InteractionStatus* is True, the Interactible is currently being interacted with by the user. As a consequence, no events of the type UserInput will be generated by the active Scene object. These events shall be handled directly by the Interactible.

At any time, at most one Interactible shall have its *InteractionStatus* set to True.

NOTES

1 The only way to change the *InteractionStatus* internal attribute is by using the SetInteractionStatus action.

2 Although Links that trigger on UserInput events cannot fire when the *InteractionStatus* internal attribute is True, other Links may still fire. This makes it possible, for instance, to implement time-outs for the interaction process.

- Boolean value.
- Initial value: False.

41.2 Events

This class defines the following events:

Interaction-Completed This event is generated as a result of a change in an Interactible. The event is generated only once per interaction, i.e., it is generated only when the *InteractionStatus* internal attribute returns to False after an interaction.

- No associated data.

HighlightOn This event is generated when the *HighlightStatus* attribute of the Interactible changes from False to True.

- No associated data.

HighlightOff This event is generated when the *HighlightStatus* attribute of the Interactible changes from True to False.

- No associated data.

CursorEnter This event is generated automatically by the MHEG-5 engine if, and only if:

1. The engine implements the Free Moving Cursor option.
2. The moving cursor has entered the area defined by the Interactible.
3. The Interactible is active.

NOTE - See subclause 53.6 for more details on the generation of *CursorLeave* and *CursorEnter* events.

CursorLeave This event is generated automatically by the MHEG-5 engine if, and only if:

1. The engine implements the Free Moving Cursor option.
2. The moving cursor has left the area defined by the Interactible.
3. The Interactible is active.

NOTE - See subclause 53.6 on main mechanisms for more details on the generation of *CursorLeave* and *CursorEnter* events.

41.3 Internal behaviours

This class defines the following internal behaviour:

Interaction Execute the following steps synchronously:

1. Set the *InteractionStatus* internal attribute to True.
2. Generate visual feedback to the user that signals the fact that the Interactible may now be interacted with.

NOTE - Most of the Interactible subclasses expand on this behaviour.

41.4 Effect of MHEG-5 actions

This class defines the following applicable MHEG-5 actions:

SetInteractionStatus (*NewInteraction-Status*) This action is used to influence the *InteractionStatus* internal attribute.

Execute the following sequence of actions:

1. If *NewInteractionStatus* is set to True,
   a) if the target Interactible or any other Interactible in the current Scene has its *InteractionStatus* attribute set to True, discard the action. Otherwise,
   b) apply the *Interaction* behaviour of the target Interactible.
2. If *NewInteractionStatus* is set to False,
   a) if the target Interactible has its *InteractionStatus* set to False, discard the action. Otherwise,
   b) immediately interrupt the *Interaction* behaviour that is taking place for the target Interactible. The state of the target Interactible after the *Interaction* behaviour is interrupted shall reflect any interaction which has taken place up to the point where the interaction was interrupted.

Provisions of use:

- The *Target* object shall be an active Interactible object.

Syntax description:

```
SetInteractionStatus    -->  Target,
                             NewInteractionStatus
Target                  -->  GenericObjectReference
NewInteractionStatus    -->  GenericBoolean
```

GetInteractionStatus (*InteractionStatusVar*)

Set the Variable referenced by *InteractionStatusVar* to the value of the *InteractionStatus* attribute.

Provisions of use:

- The *Target* object shall be available.
- *InteractionStatusVar* shall refer to an active BooleanVariable object.

Syntax description:

```
GetInteractionStatus      -->  Target,
                               InteractionStatusVar
Target                    -->  GenericObjectReference
InteractionStatusVar      -->  ObjectReference
```

SetHighlightStatus (*NewHighlightStatus*)

Change the highlighted state of an Interactible.

Execute synchronously the following sequence of actions:

1. If the current *HighlightStatus* is equal to *NewHighlightStatus*, discard the action. Otherwise,
2. Set the *HighlightStatus* attribute of the target Interactible to *NewHighlightStatus*.
3. If the target Interactible is active, if EngineResp is set to True and if *HighlightStatus* is set to True, redraw the target Interactible to present the visual feedback associated to the highlighted state.
4. If the target Interactible is active, if EngineResp is set to True and if *HighlightStatus* is set to False, redraw the target Interactible to remove the visual feedback associated to the highlighted state.

Provisions of use:

- The *Target* object shall be an available Interactible object.

Syntax description:

```
SetHighlightStatus      -->  Target,
                             NewHighlightStatus
Target                  -->  GenericObjectReference
NewHighlightStatus      -->  GenericBoolean
```

GetHighlightStatus (*HighlightStatusVar*)

Set the Variable referenced by *HighlightStatusVar* to the value of the *HighlightStatus* attribute.

Provisions of use:

- The *Target* object shall be available.
- *HighlightStatusVar* shall refer to an active BooleanVariable object.

Syntax description:

```
GetHighlightStatus        -->   Target,
                                HighlightStatusVar
Target                    -->   GenericObjectReference
HighlightStatusVar        -->   ObjectReference
```

41.5 Formal description

```
Interactible Class        -->  EngineResp?,
                               HighlightRefColour?,
EngineResp                -->  BOOLEAN
HighlightRefColour        -->  Colour
```

42 Slider Class

Description	Defines an interaction widget used to set a position within a linear range.
Base classes	Visible, Interactible
Subclasses	None
Status	Concrete class

42.1 Attributes

This subclause defines inherited, exchanged and internal attributes for this class.

42.1.1 Inherited attributes

This class has all the attributes of its base classes, with the following constraints:

Attribute Name	Defined in	Constraints and Requirements
ContentHook	Ingredient	This attribute shall not be encoded for this class.
OriginalContent	Ingredient	This attribute shall not be encoded for this class.
BoxSize, Position	Visible	In addition to defining the bounding box of the Slider, these attributes also define the actual size of the Slider. This size shall be such that the Slider completely fills its bounding box.

42.1.2 Own exchanged attributes

This class defines the following additional exchanged attributes:

Orientation — Orientation of the main axis of the Slider.

The orientation specifies the direction in which the Slider moves from the minimum towards the maximum value.

Although the exact rendering of the Slider object is not specified in detail by this part of ISO/IEC 13522, it shall be rendered such that its orientation conforms to this attribute.

- Possible values: *left* / *right* / *up* / *down*

InitialValue — The initial value of the Slider, i.e., the value it has when it is first activated after it has been prepared.

The value shall be an Integer and it shall be consistent with MinValue, MaxValue, and StepSize in the manner described below.

- Optional Integer.
- Default value: MinValue.

MinValue — Lowest value that the *SliderValue* attribute may be set to.

The value shall be consistent with InitialValue, MaxValue, and StepSize in the manner described below.

- Optional Integer.
- Default value: 1.

MaxValue

Greatest value that the *SliderValue* attribute may be set to.

The value shall be consistent with InitialValue, MinValue, and StepSize in the manner described below.

- Integer value.
- Default value: None.

InitialPortion

Represents a portion of the range [MinValue, MaxValue]. This value shall be smaller than or equal to the number (MaxValue – MinValue).

This value shall be encoded if and only if the value of the SliderStyle attribute is *proportional, in which case the following double inequality is always True:*

$$MinValue \leq InitialValue \leq MaxValue - InitialPortion$$

- Optional Integer.
- Default value: None.

StepSize

The smallest value by which the value of the *SliderValue* internal attribute may be increased or decreased.

The value shall be a positive Integer, and it shall be consistent with InitialValue, MinValue, and MaxValue in the following manner:

All values that the slider may take are expressed as

vi = MinValue + i x StepSize,

where i = 0 ... (MaxValue - MinValue)/StepSize.

The attributes InitialValue, MinValue, MaxValue and StepSize shall conform to the following criteria:

InitialValue = MinValue + M x StepSize, for some M = 0 ... (MaxValue - MinValue)/StepSize.

MinValue < MaxValue

N x StepSize = (MaxValue - MinValue), where N is some positive Integer.

- Optional Integer.
- Default value: 1.

SliderStyle

This attribute may take the value *normal*, *thermometer*, and *proportional*. The *SliderStyle* attribute influences the rendering of the Slider in the following way:

- If the *SliderStyle* is set to *normal*, the Slider is rendered as a «marker,» which is positioned on a «main axis» at the position corresponding to the *SliderValue* attribute.
- If the *SliderStyle* is set to *thermometer*, the Slider is rendered as a «main axis,» which is filled from its beginning to the position corresponding to the *SliderValue* attribute.
- If the *SliderStyle* is set to *proportional*, the Slider is rendered as a «main axis,» which is filled from the position corresponding to the *SliderValue* attribute to the position corresponding to the sum of the *SliderValue* and the *Portion* internal attributes.

This part of ISO/IEC 13522 does not specify exactly what this rendering should look like. The following picture is provided by way of example only:

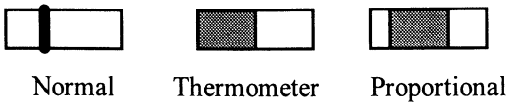


Figure 13: Examples of Sliders in different *SliderStyles*

- Optional attribute.
- Possible values: *normal* / *thermometer* / *proportional.*
- *Default value: normal.*

SliderRefColour — Specify a colour that may be used by the engine to render the Slider object.

The *SliderRefColour* value is expressed either as an absolute colour value or as a zero-based index in a colour look-up table. In the latter case, the Slider object must have the *PaletteRef* attribute encoded, which is then used to translate the index to an actual colour value.

Exactly how this colour is used to render the Slider is not specified by this part of ISO/IEC 13522. It is provided as a hint to the MHEG-5 engine on the colour scheme to use when rendering the Slider.

The actual colour resolution in the rendering process is outside the scope of this part of ISO/IEC 13522.

- Optional attribute.
- Default value: Value encoded by Application.

42.1.3 Own internal attributes

This class defines the following additional internal attributes:

SliderValue — The current value of the slider.

Portion — The current value of the attribute that governs the rendering of the Slider when the *SliderStyle* is *proportional.*

If the *SliderStyle* is set to *proportional*, the Slider is rendered as a «main axis» which is filled from the position corresponding to the *SliderValue* attribute to the position corresponding to the sum of the *SliderValue* and the *Portion* internal attributes.

42.2 Events

This class has the same events as its base class, with identical semantics:

42.3 Internal behaviour

The following internal behaviours semantics have changed from this object's base class:

Interaction — Execute the following sequence of actions:

1. Apply the Interaction behaviour as defined in the Interactible class.
2. Allow the user to interact with the Slider object by moving the marker along the main axis. Exactly how this user interaction takes place is not specified

by this part of ISO/IEC 13522. However, the smallest marker displacement shall be proportional to the value of the *StepSize* attribute.

3. When the marker has stopped at a new position,
   a) set the *SliderValue* attribute to a value that corresponds to the new marker position,
   b) set the *InteractionStatus* attribute to False, and
   c) generate an *InteractionCompleted* event.

42.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base classes, with identical semantics. In addition, the following applicable MHEG-5 actions are defined:

Step (*NbOfSteps*)

Set a new value to a slider relatively to its current value.

Execute the following sequence of actions:

1. If *NbOfSteps* is positive, increase the value of *SliderValue* by *NbOfSteps* X *StepSize*.
2. If *NbOfSteps* is negative, decrease the value of *SliderValue* by *NbOfSteps* X *StepSize*.
3. If the target Slider is active, redraw the Slider using the new value of *SliderValue* and according to its position in the *DisplayStack* of the active Application object.
4. Generate an *InteractionCompleted* event.

Provisions of use:

- The *Target* object shall be an available Slider object.
- *NbOfSteps* shall be set so that:

  MinValue <= (*NbOfSteps* X *StepSize*) + *SliderValue*, and

  (*NbOfSteps* X *StepSize*) + *SliderValue* + *Portion* <= MaxValue

  The value of *Portion* in the expressions above shall be taken to be 0 if the SliderStyle is not proportional.

Syntax description:

```
Step                --> Target,
                         NbOfSteps
Target              --> GenericObjectReference
NbOfSteps           --> GenericInteger
```

SetSliderValue (*NewSliderValue*)

Set an absolute value to a slider.

Execute the following sequence of actions:

1. Set the *SliderValue* attribute of the target Slider to *NewSliderValue*.
2. If the target Slider is active, redraw the Slider taking into account the new value of *SliderValue* and according to its position in the *DisplayStack* of the active Application object.
3. Generate an *InteractionCompleted* event.

Provisions of use:

- The *Target* object shall be an available Slider object.

  NewSliderValue shall be within the range [MinValue, MaxValue-*Portion*].

  The value of *Portion* in the expression above shall be taken to be 0 if the SliderStyle is not proportional.
- (*NewSliderValue* - MinValue) MOD StepSize shall be equal to 0.

Syntax description:

```
SetSliderValue          -->  Target,
                             NewSliderValue
Target                  -->  GenericObjectReference
NewSliderValue          -->  GenericInteger
```

GetSliderValue (*SliderValueVar*)

Set the Variable referenced by *SliderValueVar* to the value of the *SliderValue* attribute.

Provisions of use:

- The *Target* object shall be an available Slider object.
- *SliderValueVar* shall refer to an active IntegerVariable object.

Syntax description:

```
GetSliderValue          -->  Target,
                             SliderValueVar
Target                  -->  GenericObjectReference
SliderValueVar          -->  ObjectReference
```

SetPortion (*NewPortion*)

Set the size of the portion represented by a slider of style *proportional*.

Execute the following sequence of actions:

1. Set the *Portion* attribute of the target Slider to *NewPortion*.
2. If the target Slider is active, redraw the Slider according to the new value of *Portion* and according to its position in the *DisplayStack* of the active Application object.
3. Generate an *InteractionCompleted* event.

Provisions of use:

- The Target object shall be an available Slider object of style *proportional*.
- *NewPortion* shall be smaller than or equal to (MaxValue - SliderValue)

Syntax description:

```
SetPortion              -->  Target,
                             NewPortion
Target                  -->  GenericObjectReference
```

```
NewPortion              -->  GenericInteger
```

GetPortion (*PortionVar*)

Set the Variable referenced by *PortionVar* to the value of the *Portion* attribute.

Provisions of use:

- The *Target* object shall be an available Slider object of style *proportional*.
- *PortionVar* shall refer to an active IntegerVariable object.

Syntax description:

```
GetPortion              -->  Target,
                             PortionVar
Target                  -->  GenericObjectReference
PortionVar              -->  ObjectReference
```

42.5 Formal description

```
Slider Class            -->  Visible Class,
                             Interactible Class,
                             Orientation,
                             Max Value,
                             Min Value?,
                             InitialValue?,
                             InitialPortion?,
                             StepSize?,
                             SliderStyle?,
                             SliderRefColour?
Orientation             -->  left | right | up | down
InitialValue            -->  INTEGER
InitialPortion          -->  INTEGER
Min Value               -->  INTEGER
Max Value               -->  INTEGER
StepSize                -->  INTEGER
SliderStyle             -->  normal | thermometer | proportional
SliderRefColour         -->  Colour
```

43 EntryField Class

Description	Defines an interaction widget used by the final user to edit and/or modify a text.
Base classes	Text, Interactible
Subclasses	None
Status	Concrete class

43.1 Attributes

This subclause defines inherited, exchanged and internal attributes for this class.

43.1.1 Inherited attributes

This class has all the attributes of its base class, with the same semantics.

43.1.2 Own exchanged attributes

This class defines the following additional exchanged attributes:

InputType — Type of allowed characters.

If this attribute is set to *alpha*, only characters that are not digits (i.e., 0123456789) shall be accepted as entries. If this attribute is set to *numeric,* only characters that are digits shall be accepted as be entries. If this attribute is set to *any*, all characters shall be accepted. If this attribute is set to *listed*, only such characters shall be accepted that are provided in the CharList attribute of the EntryField. This provides a possibility to customise the input filter.

NOTE - The value *any* also allows for entry of symbols (e.g., &*$#@!).

- Possible values: *alpha* / *numeric* / *any* / *listed.*
- *Default value: any.*

CharList — Characters that might be entered in this EntryField.

This attribute shall always be encoded when InputType is set to *listed*. Otherwise, it shall not be encoded.

- Optional OctetString.
- Default value: None.

ObscuredInput — Indicate how to echo back input characters. This may be used for password input.

If this attribute is set to True, the entered characters shall be echoed back to the screen in an unreadable form. Otherwise, the entered characters shall be echoed back to the screen in a readable form.

- Optional Boolean value.
- Default value: False.

MaxLength — Provide the maximum number of expected input characters. When this maximum number of characters is reached in the EntryField, an EntryFieldFull event is generated.

If *MaxLength* is set to 0, the number of expected characters is undefined. In this case, the MHEG-5 engine shall provide the user with some means of terminating the interaction.

- Optional Integer.
- Default value: 0.

43.1.3 Own internal attributes

This class defines the following additional internal attributes:

EntryPoint Defines (as a zero-based index) where the next character to be entered in the EntryField shall be placed.

- If this attribute is equal to 0, the next character shall be placed before the first character of the EntryField.
- If this attribute is equal to or greater than the length of the text currently in the EntryField, the next character shall be attached to the end of the EntryField.
- For all other values n of this attribute, the next character shall be inserted after the n^{th} character of the EntryField.

- Integer value.
- Initial value: 0.

OverwriteMode Determine whether new input characters overwrite characters in the existing text or are inserted between them.

If this attribute is True, each character entered in the EntryField replaces (overwrites) the character that was previously located at the *EntryPoint*, all other characters are untouched.

If this attribute is False, the entry is inserted just before the character located at the *EntryPoint*.

If the *EntryPoint* is set to a value equal to or greater than the length of the text of the EntryField, the characters that are entered are attached to the end of the EntryField regardless of the value of this attribute.

- Boolean.
- Default value: False.

43.2 Events

This class has the same events as its base class, with identical semantics. In addition, the following events are defined:

InteractionCompleted This event is generated as a result of the ending the modification process of an EntryField.

EntryFieldFull This event is generated when an EntryField capacity is reached. This event shall only be generated when MaxLength attribute is encoded, in which case it is generated when the number of characters in the EntryField reaches MaxLength.

The event is asynchronous.

A SetData action shall not generate EntryFieldFull events.

- No associated data.

43.3 Internal behaviours

The following internal behaviours semantics have changed from this object's base class:

Interaction Execute the following sequence of actions:

1. Apply the *Interaction* behaviour as defined in the Interactible class.
2. Allow the user to modify the *TextData* attribute of the EntryField object by entering characters.
3. After each character is entered, update the *TextData* attribute accordingly (taking into account the value of the *EntryPoint* and *Overwrite* attributes), update the *EntryPoint* attribute.
4. When the entry is complete (because the user terminates the entry or because the application terminates it using the SetInteractionStatus action),
   a) set the *InteractionStatus* attribute to False,
   b) generate an *InteractionCompleted* event.

43.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base class, with identical semantics. In addition, the following applicable MHEG-5 actions are defined:

SetOverwriteMode (*NewOverwrite-Mode*) Change an entry field to toggle to and from overwriting mode.

Execute the following sequence of actions:

1. If current *OverwriteMode* is equal to *NewOverwriteMode*, discard the action. Otherwise,
2. Set *OverwriteMode* of the target EntryField to *NewOverwriteMode*.
3. If the target EntryField is active, update the visible representation of the EntryField according to the new value of *OverwriteMode*.

Provisions of use:

- The *Target* object shall be an available EntryField object.

Syntax description:

```
SetOverwriteMode      -->  Target,
                           NewOverwriteMode
Target                -->  GenericObjectReference
NewOverwriteMode      -->  GenericBoolean
```

GetOverwriteMode (*OverwriteModeVar*) Set the Variable referenced by *OverwriteModeVar* to the value of the *OverwriteMode* attribute.

Provisions of use:

- The *Target* object shall be an available EntryField object.
- *OverwriteModeVar* shall refer to an active BooleanVariable object.

Syntax description:

```
GetOverwriteMode      -->  Target,
                           OverwriteModeVar
Target                -->  GenericObjectReference
OverwriteModeVar      -->  ObjectReference
```

SetEntryPoint (*NewEntryPoint*)

Change the position of the entry point of an entry field.

Execute the following sequence of actions:

1. Set *EntryPoint* of the target EntryField to *NewEntryPoint*.
2. If the target EntryField is active, update the visible representation of the EntryField according to the new value of *EntryPoint*.

Provisions of use:

- The *Target* object shall be an available EntryField.
- *NewEntryPoint* shall be greater than or equal to 0.

Syntax description:

```
SetEntryPoint         -->  Target,
                           NewEntryPoint
Target                -->  GenericObjectReference
NewEntryPoint         -->  GenericInteger
```

GetEntryPoint (*EntryPointVar*)

Set the Variable referenced by *EntryPoinVar* to the value of the EntryPoint attribute.

Provisions of use:

- The *Target* object shall be an available EntryField object.
- *EntryPointVar* shall refer to an active IntegerVariable object.

Syntax description:

```
GetEntryPoint         -->  Target,
                           EntryPointVar
Target                -->  GenericObjectReference
EntryPointVar         -->  ObjectReference
```

43.5 Formal description

```
EntryField Class      -->  Text Class,
                           Interactible Class,
                           InputType?,
                           CharList?,
                           ObscuredInput?,
                           MaxLength?
InputType             -->  alpha | numeric | any | listed
```

```
CharList            -->  OctetString
ObscuredInput       -->  BOOLEAN
MaxLength           -->  INTEGER
```

44 HyperText Class

Description — The HyperText class is a subclass of the Text class, with the special property of allowing for portions of the text to be associated with a piece of information. These portions of text are called *anchors* in the rest of this clause.

Base classes — Text, Interactible

Subclasses — None

Status — Concrete class

44.1 Attributes

This subclause defines inherited, exchanged and internal attributes for this class.

44.1.1 Inherited attributes

This class has all the attributes of its base class, with the same semantics.

44.1.2 Own exchanged attributes

This class defines no additional exchanged attribute.

44.1.3 Own internal attributes

This class defines the following additional internal attributes:

LastAnchorFired — Tag of the last anchor fired.

- OctetString.
- Initial value: Empty string.

44.2 Events

This class has the same events as its base class, with identical semantics. In addition, the following events are defined:

AnchorFired — Signals that the user has selected one of the anchors in the HyperText object.

- Associated data: the tag of the anchor that fired - OctetString.

44.3 Internal behaviours

The following internal behaviours semantics have changed from this object's base class:

Interaction — Execute the following sequence of actions:

1. Apply the *Interaction* behaviour as inherited from the Interactible class.
2. Allow the user to move the focus through the set of anchors in the HyperText object and to select the focused anchor. Each time an anchor is selected, an *AnchorFired* event is generated.
3. When the interaction is complete (either because the user terminates the

interaction or because the application terminates it using the SetInteractionStatus action),

a) set the *InteractionStatus* attribute to False,

b) generate an *InteractionCompleted* event.

44.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base class, with identical semantics. In addition, the following applicable MHEG-5 actions are defined:

GetLastAnchorFired (*LastAnchor-FiredVar*)

Set the Variable referenced by *LastAnchorFiredVar* to the value of the *LastAnchorFired* attribute.

Provisions of use:

- The *Target* object shall be an available Hypertext object.
- *LastAnchorFired* shall refer to an active OctetStringVariable object.

Syntax description:

```
GetLastAnchorFired    -->  Target,
                           LastAnchorFiredVar
Target                -->  GenericObjectReference
LastAnchorFiredVar    -->  ObjectReference
```

44.5 Formal description

```
HyperText Class          -->  Text Class,
                              Interactible Class
```

45 Button Class

Description	Defines functionality associated with the rendering and interaction with one-state and two-state buttons.
Base class	Visible, Interactible
Subclasses	Hotspot, PushButton
Status	Abstract class

45.1 Attributes

This subclause defines inherited, exchanged and internal attributes for this class.

45.1.1 Inherited attributes

This class has all the attributes of its base classes, with the following constraints:

Attribute Name	Defined in	Constraints and Requirements
ContentHook	Ingredient	This attribute shall not be encoded for this class.
OriginalContent	Ingredient	This attribute shall not be encoded for this class.
InteractionStatus	Interactible	This internal attribute is not defined for the Button class.
BoxSize, Position	Visible	In addition to defining the bounding box of the Button, these attributes also define the actual size of the Button. This size shall be such that the Button completely fills its bounding box.

45.1.2 Own exchanged attributes

This class defines the following additional exchanged attributes:

ButtonRefColour

Specify a colour that may be used by the engine to render the Button object.

The *ButtonRefColour* attribute value is expressed either as an absolute colour value or as a zero-based index in a colour look-up table. In the latter case, the Button object must have the *PaletteRef* attribute encoded, which is then used to translate the index to an actual colour value.

Exactly how this colour is used to render the Button is not specified by this part of ISO/IEC 13522. It is provided as a hint to the MHEG-5 engine on the colour scheme to use when rendering the Button.

The actual colour resolution in the rendering process is outside the scope of this part of ISO/IEC 13522.

- Optional attribute.
- Default value: Value encoded by Application.

45.1.3 Own internal attributes

This class defines the following additional internal attributes:

SelectionStatus

Each button may store one bit of information. This attribute is True when the Button object is in the selected state. The selected state is entered as a result of invoking the Select action.

- Boolean value.
- Initial value: False

45.2 Events

IsSelected This event is generated when the *SelectionStatus* of the Button changes from False to True.

- No associated data.

IsDeselected This event is generated when the *SelectionStatus* of the Button changes from True to False.

- No associated data.

45.3 Internal behaviours

The following internal behaviours semantics have changed from this object's base class:

Interaction This behaviour is not defined for the Button class.

NOTE - As a result, the Button class shall not generate any InteractionCompleted event.

Selection Execute the following sequence of actions:

1. Set the *SelectionStatus* to True.
2. If the *EngineResp* attribute is True and the Button is active, redraw the Button object taking into account the new value of SelectionStatus and according to its position in the *DisplayStack* of the active Application object.
3. Generate an *IsSelected* event.

Deselection Execute the following sequence of actions:

1. Set the *SelectionStatus* to False.
2. If the *EngineResp* attribute is True and the Button is active, redraw the Button object taking into account the new value of SelectionStatus and according to its position in the *DisplayStack* of the active Application object.
3. Generate an *IsDeselected* event.

45.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base class, with identical semantics. In addition, the following applicable MHEG-5 actions are defined:

SetInteractionStatus This action shall not be targeted to a Button object.

GetInteractionStatus This action shall not be targeted to a Button object.

Select Execute the following sequence of actions:

1. If *SelectionStatus* is currently set to True, disregard this action.
2. If *SelectionStatus* is currently set to False, apply the *Selection* behaviour of the

target Button.

Provisions of use:

- The *Target* object shall be an available Button.

Syntax description:

```
Select                  -->  Target
Target                  -->  GenericObjectReference
```

Deselect Execute the following sequence of actions:

1. If *SelectionStatus* is currently set to False, disregard this action.
2. If *SelectionStatus* is currently set to True, apply the *Deselection* behaviour of the target Button.

Provisions of use:

- The *Target* object shall be an available Button.

Syntax description:

```
Deselect                -->  Target
Target                  -->  GenericObjectReference
```

45.5 Formal description

```
Button Class            -->  Visible Class,
                             Interactible Class,
                             ButtonRefColour?
ButtonRefColour         -->  Colour
```

46 Hotspot Class

Description	Defines invisible unlabelled rectangular areas on the screen that may interact with the user to produce *IsSelected* events.
Base class	Button
Subclasses	None
Status	Concrete class

46.1 Attributes

This subclause defines inherited, exchanged and internal attributes for this class.

46.1.1 Inherited attributes

This class has all the attributes of its base classes, with identical semantics, except for the following attribute:

SelectionStatus Rendering of a Hotspot object shall depend on the *SelectionStatus* attribute. When the *SelectionStatus* attribute and the EngineResp are both True, the Hotspot shall be rendered in a way that signals to the user that selection has taken place. For this rendering process, the attribute ButtonColour may be used. In all other cases, the Hotspot shall have no visual rendering except such rendering as is prescribed by the HighlightStatus of its base class.

46.1.2 Own exchanged attributes

This class defines no additional exchanged attribute.

46.1.3 Own internal attributes

This class defines no additional internal attribute.

46.2 Events

This class has the same events as its base classes, with identical semantics.

46.3 Internal behaviours

This class has the same internal behaviours as its base class, with identical semantics.

46.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base class. The following MHEG-5 action semantics have changed from this object's base class:

Select Execute the following sequence of actions:

1. Apply the *Selection* behaviour as defined in the base class.
2. Apply the *Deselection* behaviour as defined in the base class.

The provisions of use and syntax description of the action are unchanged.

46.5 Formal description

```
Hotspot Class          -->  Button Class
```

47 PushButton Class

Description	Defines labelled, largely rectangular areas on the screen that may interact with the user to produce *IsSelected* events.
Base class	Button
Subclasses	SwitchButton
Status	Concrete class

47.1 Attributes

This subclause defines inherited, exchanged and internal attributes for this class.

47.1.1 Inherited attributes

This class has all the attributes of its base classes, with identical semantics, except for the following attribute:

SelectionStatus Rendering of a PushButton object shall depend on the *SelectionStatus* attribute. When the *SelectionStatus* attribute and the EngineResp attribute are both True, the PushButton shall be rendered in a way that signals to the user that selection has taken place. This rendering shall depict a button that has been pressed. In all other cases, the Button shall be rendered in a way that signals to the user that selection has not taken place. This rendering shall depict a button that has not been pressed.

47.1.2 Own exchanged attributes

This class defines the following additional exchanged attributes:

OriginalLabel One-line piece of text that represents the initial label of the PushButton.

- Optional OctetString.
- Default value: Empty string.

CharacterSet Identification of the character set, or set of character sets, that shall be used by default for rendering of the label. This Integer shall be encoded with a value representing the character set. The application domain shall define a range of CharacterSet and its semantics.

NOTE - The *CharacterSet* attribute provides the initial character set for a label. In addition, the text encoding format may contain escape sequences to switch between character sets.

- Optional Integer.
- Default value: The value of *CharacterSet* attribute from the Application object, if that attribute is specified.

47.1.3 Own internal attributes

This class defines the following additional internal attributes:

Label Label of the PushButton.

- Optional OctetString.

- Initial value: Value of the OriginalLabel attribute.

47.2 Events

This class has the same events as its base class, with identical semantics.

47.3 Internal behaviours

This class has all the internal behaviours of its base class, with identical semantics.

47.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base class, with the following changes and extensions:

Select — Execute the following sequence of actions:

1. Apply the *Selection* behaviour as defined in the base class.
2. Apply the *Deselection* behaviour as defined in the base class.

The provisions of use and syntax description of the action are unchanged.

SetLabel (*NewLabel*) — Change the label of a PushButton.

Execute the following sequence of actions:

1. Set *Label* attribute of the target PushButton to *NewLabel*.
2. If the target PushButton is active, redraw the PushButton taking into account the new value of *Label* and according to its position in the *DisplayStack* of the active Application object.

Provisions of use:

- The *Target* object shall be an available PushButton.

Syntax definition:

```
SetLabel                --> Target,
                            NewLabel

Target                  --> GenericObjectReference
NewLabel                --> GenericOctetString
```

GetLabel (*LabelVar*) — Set the Variable referenced by *LabelVar* to the value of the *Label* attribute.

Provisions of use:

- The *Target* object shall be an available PushButton.
- *LabelVar* shall refer to an active OctetStringVariable object.

Syntax description:

```
GetLabel                --> Target,
                            LabelVar
```

```
Target                    -->  GenericObjectReference
LabelVar                  -->  ObjectReference
```

47.5 Formal description

```
PushButton Class          -->  Button Class,
                               OriginalLabel?,
                               CharacterSet?
OriginalLabel             -->  OctetString
```

48 SwitchButton Class

Description	Defines a data structure to deal with labelled, largely rectangular areas on the screen that may interact with the user to produce *IsSelected* and *IsDeselected* events.
Base class	PushButton
Subclasses	None
Status	Concrete class

48.1 Attributes

This class has all the attributes of its base classes, with identical semantics, except for the following attribute:

SelectionStatus Rendering of a SwitchButton object shall depend on the *SelectionStatus* attribute.

When the *SelectionStatus* attribute and the *EngineResp* are both True, the SwitchButton shall be rendered in a way that signals to the user that selection has taken place. This rendering shall depict a radio button or a checkbox that has been selected, or a push button that has been pressed, depending on the *ButtonStyle* attribute. In all other cases, the Button shall be rendered in a way that signals to the user that selection has not taken place. This rendering shall depict a radio button or a checkbox that has not been selected, or a button that has not been pressed.

48.1.1 Inherited attributes

This class has all the attributes of its base classes, with identical semantics.

48.1.2 Own exchanged attributes

This class defines the following additional exchanged attributes:

ButtonStyle Presentation style of the SwitchButton.

- Possible values: *pushbutton* | *radiobutton* | *checkbox*

48.1.3 Own internal attributes

This class defines no additional internal attributes.

48.2 Events

This class has the same events as its base class, with identical semantics.

48.3 Internal behaviours

This class has the same internal behaviours as its base class, with identical semantics.

48.4 Effect of MHEG-5 actions

This class has the same set of MHEG-5 actions as its base class, with identical semantics. In addition, the following applicable MHEG-5 actions are defined:

GetSelectionStatus (*SelectionStatusVar*)

Return the value of the *SelectionStatus* attribute in the form of a Boolean in the Variable referenced by the *SelectionStatusVar* parameter.

Provisions of use:

- The *Target* object shall be an available SwitchButton object.
- *SelectionStatusVar* shall refer to an active BooleanVariable object.

Syntax description:

```
GetSelectionStatus    -->  Target,
                           SelectionStatusVar
Target                -->  GenericObjectReference
SelectionStatusVar    -->  ObjectReference
```

Select

If the *SelectionStatus* attribute is True, disregard this action. Otherwise, invoke the *Selection* behaviour.

Deselect

If the *SelectionStatus* attribute is False, disregard this action. Otherwise, invoke the *Deselection* behaviour.

Provision of use:

- The *Target* object shall be an available SwitchButton object.

Syntax description:

```
Deselect              -->  Target
Target                -->  GenericObjectReference
```

Toggle

If the *SelectionStatus* attribute is False, invoke the *Selection* behaviour. Otherwise, invoke the *Deselection* behaviour.

Provision of use:

- The *Target* object shall be an available SwitchButton object.

Syntax description:

```
Toggle                -->  Target
Target                -->  GenericObjectReference
```

SetLabel

Execute the following sequence of actions:

1. Set the *Label* attribute to the new value.
2. If the *Target* SwitchButton is active, redraw the SwitchButton taking into account the new value of *Label* attribute and according to its position on the DisplayStack of the active Application object.

Provision of use:

- The *Target* object shall be an available SwitchButton object.

The syntax of this action is defined in the PushButton class.

48.5 Formal description

```
SwitchButton Class      -->  PushButton Class,
                             ButtonStyle
ButtonStyle             -->  pushbutton   |   radiobutton   |
                             checkbox
```

49 Action Class

Description	The Action class is a list of elementary actions that are intended to be executed synchronously. The Action class does not inherit from any MHEG-5 class; as a result, Action objects cannot be referenced individually.
Base class	None
Subclasses	None
Status	Concrete class

49.1 Attributes

This subclause defines the inherited, exchanged and internal attributes of this class.

49.1.1 Inherited attributes

This class has no inherited attributes.

49.1.2 Own exchanged attributes

This class defines the following exchanged attribute:

ElementaryActions	List of included elementary actions

49.2 Own internal attributes

This class defines no internal attribute.

49.3 Formal description

```
Action Class       --> ElementaryAction+
ElementaryAction   --> Activate | Add | AddItem | Append | BringToFront |
                       Call | CallActionSlot | Clear | Clone |
                       CloseConnection | Deactivate | DelItem | Deselect |
                       DeselectItem | Divide | DrawArc | DrawLine | DrawOval
                       | DrawPolygon | DrawPolyline | DrawRectangle
                       |DrawSector | Fork | GetAvailabilityStatus |
                       GetBoxSize | GetCellItem | GetCursorPosition |
                       GetEngineSupport | GetEntryPoint | GetFillColour |
                       GetFirstItem | GetHighlightStatus |
                       GetInteractionStatus | GetItemStatus | GetLabel |
                       GetLastAnchorFired | GetLineColour | GetLineStyle |
                       GetLineWidth | GetListItem | GetListSize |
                       GetOverwriteMode | GetPortion | GetPosition |
                       GetRunningStatus | GetSelectionStatus | GetSliderValue
                       | GetTextContent | GetTextData | GetTokenPosition |
                       GetVolume | Launch | LockScreen | Modulo | Move |
                       MoveTo | Multiply | OpenConnection | Preload |
                       PutBefore | PutBehind | Quit | ReadPersistent | Run |
                       ScaleBitmap | ScaleVideo | ScrollItems | Select |
                       SelectItem | SendEvent | SendToBack | SetBoxSize |
                       SetCachePriority | SetCounterEndPosition |
                       SetCounterPosition | SetCounterTrigger |
                       SetCursorPosition | SetCursorShape | SetData |
```

	SetEntryPoint | SetFillColour | SetFirstItem | SetFontRef | SetHighlightStatus | SetInteractionStatus | SetLabel | SetLineColour | SetLineStyle | SetLineWidth | SetOverwriteMode | SetPaletteRef | SetPortion | SetPosition | SetSliderValue | SetSpeed | SetTimer | SetTransparency | SetVariable | SetVolume | Spawn | Step | Stop | StorePersistent | Subtract | TestVariable | Toggle | ToggleItem | TransitionTo | Unload | UnlockScreen

NOTE - The semantics and syntax of the elementary actions are provided earlier in the document.

50 Referencing Objects, Content, Values, Colour and XYPosition

50.1 ObjectReference

Description This data type is used for referencing objects. The object referenced shall be visible to the object from which the reference is made. This means that the object shall be either an Application or Scene object, or shall be part of either the active Application or the active Scene.

The reference consists of an optional GroupIdentifier and an ObjectNumber. The default value for the GroupIdentifier is the GroupIdentifier of the Scene or Application object from which the reference was made.

```
ObjectReference          --> GroupIdentifier?,
                             ObjectNumber
GroupIdentifier          --> OctetString
ObjectNumber             --> INTEGER
```

50.2 ContentReference

Description This data type is used for referencing external sources of data. The ContentReference consists of an OctetString.

```
ContentReference         --> OctetString
```

50.3 GenericObjectReference

Description Data type that allows either direct reference to an object or indirect reference via a Variable object.

In the case of direct reference, this reference resolves to an ObjectReference directly to the target object.

In the case of indirect reference, this reference resolves to an ObjectReference to an ObjectRefVariable object. That ObjectRefVariable object shall then contain either an ObjectReference to the target object or NULL.

```
GenericObjectReference   --> DirectReference |
                             IndirectReference
DirectReference          --> ObjectReference
IndirectReference        --> ObjectReference
```

50.4 GenericContentReference

Description Data type that allows either direct reference to an external source of data or indirect reference via a Variable object.

In the case of direct reference, this reference resolves to a ContentReference directly.

In the case of indirect reference, this reference resolves to an ObjectReference to a ContentRefVariable object. That ContentRefVariable object shall then contain either a ContentReference to the content or NULL.

```
GenericContentReference     --> DirectContentReference |
                                IndirectReference
DirectContentReference      --> ContentReference
IndirectReference           --> ObjectReference
```

50.5 GenericInteger

Description Data type that allows either direct inclusion of an Integer or reference to an IntegerVariable object.

```
GenericInteger              --> Value | IndirectReference
Value                       --> INTEGER
IndirectReference           --> ObjectReference
```

50.6 GenericBoolean

Description Data type that allows either direct inclusion of a Boolean or reference to a BooleanVariable object.

```
GenericBoolean              --> Value | IndirectReference
Value                       --> BOOLEAN
IndirectReference           --> ObjectReference
```

50.7 GenericOctetString

Description Data type that allows either direct inclusion of an OctetString or reference to an OctetStringVariable object.

```
GenericOctetString          --> Value | IndirectReference
Value                       --> OctetString
IndirectReference           --> ObjectReference
```

50.8 Colour

Description Data type used to specify a colour by a name - an OctetString- or by an index -an Integer referencing a palette object-.

```
Colour                      --> ColourIndex | AbsoluteColour
ColourIndex                 --> INTEGER
AbsoluteColour              --> OctetString
```

50.9 XYPosition

Description Data type used to specify the (X,Y) position in a Scene coordinate system.

```
XYPosition                  --> XPosition,
                                YPosition
XPosition                   --> INTEGER
YPosition                   --> INTEGER
```

50.10 Resolution of generic values

Generic values (GenericContentReference, GenericObjectReference, GenericInteger, GenericBoolean, and GenericOctetString) are used only as parameters to elementary actions. The resolution takes place when the action is invoked. As an example, consider an IntegerVariable V that is initially set to 10. If the following actions are invoked:

1. set V to 15,
2. set a Slider's *Value* to V,
3. set V to 20,

the Slider's *Value* will be set to 15. In other words, the Slider's value is set to by value, not by reference.

51 Referencing MHEG-5 Objects

References to MHEG-5 objects are represented by ObjectReferences. At any time, an ObjectReference shall be resolved by taking into account both the MHEG-5 Group object that contains this reference (a Scene or an Application) and the content of the GroupIdentifier and ObjectNumber attributes of the ObjectReference.

The following presents how the ObjectReference shall be encoded according to the Group of origin and the nature of the referenced MHEG-5 object:

I. Within a Scene object,

A. Reference to an Ingredient of the active Scene:

1. The GroupIdentifier may or may not be encoded.
2. The ObjectNumber shall contain the object number of the Ingredient within the Scene.

B. Reference to a shared Ingredient of the active Application:

1. The GroupIdentifier shall contain the group identifier of the active Application object.
2. The ObjectNumber shall contain the object number of the Ingredient within the Application object.

C. Reference to the active Scene itself:

1. The GroupIdentifier need not be encoded.
2. The ObjectNumber shall be set to 0.

D. Reference to another Scene:

1. The GroupIdentifier shall contain the group identifier of the Referenced Scene object.
2. The ObjectNumber shall be set to 0.

E. Reference to the active Application:

1. The GroupIdentifier shall contain the group identifier of the active Application object.
2. The ObjectNumber shall be set to 0.

F. Reference to another Application object:

1. The GroupIdentifier shall contain the group identifier of the referenced Application object.
2. The ObjectNumber shall be set to 0.

II. Within an Application object,

A. Reference to an Ingredient of the Application group:

1. The GroupIdentifier may or may not be encoded.
2. The ObjectNumber shall contain the unique object number of the Ingredient within the Application.

B. Reference to a Scene:

1. The GroupIdentifier shall contain the group identifier of the referenced Scene object.
2. The ObjectNumber shall be set to 0.

C. Reference to the Application itself:

1. The GroupIdentifier need not be encoded.
2. The ObjectNumber shall be set to 0.

D. Reference to another Application object:

1. The GroupIdentifier shall contain the group identifier of the referenced Application object.
2. The ObjectNumber shall be set to 0.

This part of ISO/IEC 13522 does not define the actual encoding of the GroupIdentifier OctetString. Every MHEG-5 application domain shall define specific forms of the GroupIdentifier.

52 Name Spaces, RemoteProgram Calls and Connections

The MHEG-5 engine has a default name space, which is the name space of the active Application object. All other Application objects that can be reached from it (by Launch or Spawn) shall also be in that name space. This name space, in a wide sense, also should include enough information to perform remote calls to named methods through the MHEG-5 Program objects.

NOTE - In the Annex C (informative), an explanation may be found on how the MHEG-5 engine attaches to this name space.

However, it is also possible for an MHEG-5 application to connect to another name space temporarily, using the OpenConnection action. That new name space may be used for named method calls (through the MHEG-5 Program objects) and in order to access an MHEG-5 Scene object that is located in another name space than its Application object. The following rule applies to the references to and from such a Scene object:

1. The reference to a Scene object in another name space than the Application object shall be encoded within a TransitionTo action of which the ConnectionTag parameter indicates the connection with the entity that administers that other name space.
2. All ContentReferences from that Scene object shall be interpreted in the «other» name space.
3. All GroupIdentifier references from that Scene object shall be interpreted in the default name space, except as indicated by rule 1. above.

Reminder: whether or not the MHEG-5 engine shall implement the OpenConnection and CloseConnection actions is a decision of the application domain.

53 Event Handling

53.1 Types of events

This part of ISO/IEC 13522 defines the following events: *IsAvailable, ContentAvailable, IsDeleted, IsRunning, IsStopped, TestEvent, UserInput, TimerFired, AsynchStopped, InteractionCompleted, TokenMovedFrom, TokenMovedTo, FirstItemPresented, LastItemPresented, HeadItems, TailItems, ItemSelected, ItemDeselected, StreamEvent, StreamPlaying, StreamStopped, CounterTrigger, HighlightOn, HighlightOff, CursorEnter, CursorLeave, AnchorFired, IsSelected, IsDeselected, EntryFieldFull, EngineEvent.*

An event always emanates from one specific object, called the event source. The semantics of each MHEG-5 class define the circumstances under which an object of that class generates a specific event.

Some of the event types above have associated with them a data value. That value is used in order to determine whether the associated Link should fire, as described below. The following table lists the data value associated with each event type:

Event Type	Associated data	Associated value type
AnchorFired	*AnchorTag*	*OctetString*
CounterTrigger	*Identifier*	*INTEGER*
EngineEvent	*EventTag*	*INTEGER*
FirstItemPresented	*Index*	*BOOLEAN*
HeadItems	*Number of items*	*INTEGER*
ItemDeselected	*Index*	*INTEGER*
ItemSelected	*Index*	*INTEGER*
LastItemPresented	*Index*	*BOOLEAN*
StreamEvent	*StreamEventTag*	*OctetString*
TailItems	*Number of items*	*INTEGER*
TestEvent	*TestResult*	*BOOLEAN*
TimerFired	*TimerIdentifier*	*INTEGER*
TokenMovedFrom	*Index*	*INTEGER*
TokenMovedTo	*Index*	*INTEGER*
UserInput	*UserInputEventTag*	*INTEGER*
All others	None.	N/A

Events which types have associated data must always be generated with that data. E.g., when a *TimerFired* event is generated, it must be accompanied by the *TimerIdentifier*.

53.2 Synchronous events and asynchronous events

Events may occur for two reasons.

1. A process that is asynchronous to the MHEG-5 engine produces an event.

   The resulting event is then called asynchronous. Asynchronous events are all events of the types *AnchorFired, AsynchStopped, ContentAvailable, CounterTrigger, CursorEnter, CursorLeave., EngineEvent, EntryFieldFull, InteractionCompleted, StreamEvent, StreamPlaying, StreamStopped, TimerFired and UserInput.*

2. The event is the direct result of the execution of an elementary action.

   The resulting event is then called synchronous. Synchronous events are all events of the types *FirstItemPresented, HeadItems, HighlightOff, HighlightOn, IsAvailable, IsDeleted, IsDeselected, IsRunning,*

IsSelected, IsStopped, ItemDeselected, ItemSelected, LastItemPresented, TailItems, TestEvent, TokenMovedFrom and TokenMovedTo. The semantics of the classes of this part of ISO/IEC 13522 explicitly state when such an event is generated.

53.3 Event handling and Links

Every MHEG-5 Link has a LinkCondition and a LinkEffect. When the MHEG-5 engine examines an event that has occurred, it shall check all active Links (of the active Application and Scene objects) to see if their EventType and EventSource attributes match the type and source of the event in question. For each of the Links that fulfil this condition, the associated data of the event is checked against the optional EventData attribute of the Link. Links that fulfil this condition as well (or that fulfil it by default since they have no EventData) are said to be *fired.*

The MHEG-5 engine is driven by the occurrence of asynchronous events. At the occurrence of an asynchronous event, the MHEG-5 engine shall examine all active Links of the active Application and Scene objects to determine if they have fired. For each of the fired Links, the elementary actions of its LinkEffect shall be stored in a queue for sequential execution. This part of ISO/IEC 13522 does *not* specify the order in which two Links that fire on the same event are to be handled.

Default error handling is the following: if one of the aforementioned elementary action produces an error, that elementary action shall be ignored. It is however allowed for an application domain to use EngineEvent events to indicate the error situation to the MHEG-5 application (e.g., to send to an error message).

As a direct result of a LinkEffect being executed, synchronous events may occur. These events shall be dealt with directly by the MHEG-5 engine. In other words, after the execution of each elementary action, the MHEG-5 engine shall check if any additional Links have fired as the result of a synchronous events occurring. If this is the case, that Link and all its effects shall be completely processed before the MHEG-5 engine continues to process the next elementary action of the original Link.

Any asynchronous events (such as a UserInput event) that occur while the original asynchronous events are being processed are not dealt with until after the completion of the entire process above. The asynchronous events which are not handled, are queued.

Actions that change the context of the current action processing will influence both the queue of asynchronous events and the queue of actions waiting for processing. The context is changed by "TransitionTo", "Launch", "Spawn" and "Quit" actions. If such an action occurs, the pending asynchronous events whose origin is after the context switch shall be removed from the asynchronous event queue. Elementary actions waiting for execution shall be removed from the action queue as well.

In this context, it should be noted that some MHEG-5 actions, such as Run, have an effect that continues after the completion of the action itself. For example, when a Bitmap object is run, the Run action returns as soon as the Bitmap has been displayed, thus allowing the Link processor to continue its work. The effect of the action (i.e., that the bitmap is on the screen) still continues after the completion of the action.

Another important aspect is that it is possible for a Link to deactivate itself in its LinkEffect. Such an action shall be postponed until the LinkEffect has been completely executed.

53.4 User input

This part of ISO/IEC 13522 specifies neither the user input devices nor the types of events that are generated. The «raw» user input received by the MHEG-5 engine (for instance in the form of remote control commands) shall be translated by the engine, where appropriate, into occurrences of *UserInput* events. These events all have a tag, which specifies which *UserInput* event has occurred. The tag is an Integer; the semantics of the tag are defined by a InputEventRegister attached to the Scene object.

Once the MHEG-5 engine has translated the raw user input into one or more *UserInput* events, that or those events are treated as asynchronous events by the engine as described above.

53.5 User interaction

MHEG-5 objects that belong to the class Interactible may be in a certain state, called «interacting», which is signalled by the *InteractionStatus* attribute of the object being True. When an object is in this state, no *UserInput* events shall be generated by the MHEG-5 engine. The reason for this rule is that the MHEG-5 engine might need to use the «raw» user input in order to interact with the Interactible, and therefore may not be able to generate the normal UserInput events.

At all times, at most one object shall be interacting.

However, all other events are still generated. This makes it possible, for instance, to implement time-outs.

53.6 Cursor Events

CursorLeave and *CursorEnter* events defined earlier need the following precision. It may occur that a cursor moves from one area defined by an active Interactible immediately to another area defined by another active Interactible; that may be the case, for instance, when one area is on top of the other, that is when one area is above the other in the DisplayStack and they overlap. In such case, a *CursorLeave* event shall be generated by the former Interactible (the topmost of the two in the DisplayStack), followed by a *CursorEnter* event generated by the latter Interactible (the lower one in the DisplayStack).

Cursor Events have also to be generated if an active Interactible is moved beneath the cursor position, or if an Interactible beneath the cursor position becomes active, or if the stacking order of Interactibles beneath the cursor position changes in such a way, that the topmost Interactible beneath the cursor position is changed.

In general, only one active Interactible per time can have the cursor in its bounding box. This has to be reflected by appropriate sequences of CursorLeave and CursorEnter events.

Finally, it should also be recalled here that an inactive Interactible shall not generate any cursor events.

53.7 Error Handling

In general, default error mechanism of the MHEG-5 engine consists in ignoring the cause of the error and continuing to the next step. For instance if an interchanged object is an instance of a class that the MHEG-5 engine does not recognise, the object shall be ignored. Any subsequent action that may occur in relation with this object and that produces an error shall be ignored. More generally, any cause of error (e.g. in an elementary action) shall result in ignoring that cause (e.g. the elementary action).

However, as mentioned earlier, it is allowed for an application domain to use EngineEvent events to indicate the error situation to the MHEG-5 application (e.g., to send to an error message).

54 Rendering Visibles

54.1 Coordinate system

This subclause describes the semantics associated with displaying objects of classes that inherit from the Visible class. The MHEG-5 engine shall have access to exactly one display, which is called the «screen» in the rest of this subclause. The screen is an orthogonal coordinate system, with an x axis from left to right and a y axis from top to bottom. The origin is in the top left corner. Depending on the SceneCoordinateSystem and AspectRatio attributes of the currently active Scene object, the coordinate system may have different sizes and aspect ratios.

54.2 Bounding box

Each Visible has a bounding box, specified by the internal attributes Position and BoxSize. These are both expressed in the Scene coordinate space. The contents of the Visible shall be anchored to the top left corner

of the bounding box. Furthermore, the contents of the Visible shall be clipped so that no parts that fall outside the bounding box are rendered. The clipping shall be performed in the following fashion:

- For Text and its subclasses, the clipping is performed in such a way that the positioning of the text within the bounding box is respected. Here, only the case where LineOrientation is *horizontal* is explained; the *vertical* case is obtained analogously.
  - Horizontal justification:

  If the Text object has a TextWrapping attribute set to True, its content is split into as many lines as needed in order to fit into the bounding box. In addition, if the HorizontalJustification attribute of the Text object is set to *justified*, the lines are displayed so that their beginning is aligned with the left edge of the bounding box and that their end is aligned with the right edge of the bounding box.

  If the TextWrapping attribute is set to False, the following rules apply for horizontal justification:

  If HorizontalJustification is set to *start*, the lines are clipped so that the beginning of each line is aligned with the left edge of the bounding box.

  If HorizontalJustification is set to *end*, the lines are clipped so that the end of each line is aligned with the right edge of the bounding box.

  If HorizontalJustification is set to *centre*, the lines are clipped on both sides, so that the centre of each line is aligned with a virtual vertical line at the centre of the bounding box.

  - Vertical Justification:

  For a Text that has the VerticalJustification attribute set to *end*, its content is clipped so that the end of the text is aligned with the bottom of the bounding box; thus, the beginning of the text is obscured. Conversely, if VerticalJustification is set to *start*, the text is clipped so that its end is obscured. Finally, if VerticalJustification is set to *centre*, is clipped so that the centre of the text is aligned with the centre of the bounding box, meaning that both beginning and end are obscured.

  - Background Colour:

  If the text object has a background colour other than transparent, its entire bounding box shall be filled with that colour, whether or not the text fills the bounding box.

- For Slider, Rectangle, and subclasses of Button, the object is not clipped but rather scaled to fit exactly within its bounding box.
- Other Visibles are clipped so that their top left corner is positioned at the top left corner of the bounding box. For these Visibles, some part of the bounding box may be empty; that part shall then be transparent.

54.3 Display stack

The MHEG-5 engine shall implement a display stack. Each Visible is initially rendered according to its position in the *DisplayStack* attribute of the current Application object. Visibles may be moved up and down in the display stack using the actions BringToFront, SendToBack, PutBefore and PutBehind. When a Visible is moved in this way, (parts of) other Visibles may be obscured or uncovered. The display shall then be updated accordingly.

54.4 Transparent objects

Transparency of objects are specified with an Integer in the range [0, 100] (percent). Interpreting the integer between 1 and 99 depends on the implementation and this part of ISO/IEC 13522 does not define it. Neither the actual rendering and display accuracy, nor the algorithms used to perform these tasks are specified by this part of ISO/IEC 13522. However, transparency values of 0% and 100% are specified; they are defined as follows.

Objects of 0% transparency shall be rendered as non transparent objects, i.e., the original colours shall be rendered without any modification. Objects of 100% transparency shall be rendered as completely transparent objects, i.e., the original colours shall not be rendered, and the background shall be perceived.

I. For each location in the Scene coordinate space, at most the N objects whose bounding boxes contain that location are considered. These objects are ordered in the so-called display stack. Let's call the objects O_1, O_2, [...], O_N, where O_N denotes the topmost object.

II. The computation of the final pixel value is done from the bottom of the stack. The pixel value after taking into account j objects is called V_j. The final pixel value, therefore, is called V_N.

III. The computation of the pixel value starts from a completely black background. Therefore, V_0 = <black>.

IV. For a j taking successively the values 1 through N, the following is performed:

A. If the pixel value of O_j to be considered is called P_j and is p percent translucent:

$V_j = V_{j-1} (p/100) + P_j (1 - p/100);$

This formula may be applied to all colour components of a pixel if the colour space is not based on the luminance and colour difference signals (RGB is such a case). However, this formula must be applied only to the luminance of a pixel if the colour space is based on the luminance and colour difference signals such as YUV. In addition, the values in this formula are all assumed to be expressed in one and the same linear colour space. If a non linear colour space is used, this formula may be taken into account for the precise rendering.

B. specifically, if the pixel is completely transparent (invisible):

$V_j = V_{j-1};$

C. specifically, if the pixel is completely opaque:

$V_j = P_j .$

This model shall not be understood to mandate any specific rendering and/or display process in a specific implementation of an MHEG-5 engine. Its sole purpose is to define unambiguously what the semantics are of combining different transparent graphics objects. Neither the actual rendering and display accuracy, nor the algorithms used to perform these tasks are defined by this part of ISO/IEC 13522.

54.5 Pixel aspect ratio

Generally the pixel aspect ration of a Scene and its content are assumed to be the same, a pixel of the Scene's content is then exactly mapped to a pixel of the Scene, no scaling is needed. If however the two aspect ratios are different, the MHEG-5 engine may scale the content - taking into account the two aspect ratios - in order to compensate for the presentation of the content. Note that this scaling is up to the implementation of the MHEG-5 engine and is an optional feature; this part of ISO/IEC 13522 does not define any specific way to do it.

Annex A

(normative)

ASN.1 notation

This annex describes the ASN.1 notation for the syntax of MHEG-5 objects conforming to ISO/IEC 13522-5.

The encoding of the MHEG-5 objects from this ASN.1 syntax shall make use of the *Distinguished Encoding Rules (DER)* defined in ISO/IEC 8825-1:1995 | ITU-T Recommendation X.690. The alternative encoding is the textual notation defined in Annex B.

The syntax that shall be used in the DER is detailed below.

```
--$PREFIX=ISOMHEG-mheg-5:mheg-5
-- Module: mheg-5

ISO13522-MHEG-5
{joint-iso-itu-t(2) mheg(19) version(1) mheg-5(17)} DEFINITIONS IMPLICIT
TAGS ::=
BEGIN

-- This module defines the MHEG-5 abstract syntax which consists of data values of type:
--   ISO13522-MHEG-5.InterchangedObject.
-- This abstract syntax is identified by the name: {joint-iso-itu-t(2) mheg(19) version(1) mheg-5(17)}.

InterchangedObject ::= CHOICE
{
  application [0] ApplicationClass,
  scene [1] SceneClass
}

-- A.1 Root Class ____________________________________________________

RootClass ::= ObjectReference

-- A.2 Group Class ___________________________________________________

GroupClass ::= SET
{
  RootClass (WITH COMPONENTS
    {external-reference (WITH COMPONENTS {..., object-number (0)}) PRESENT,
     internal-reference ABSENT}),
  standard-identifier [2] StandardIdentifier OPTIONAL,
  standard-version [3] INTEGER (1) OPTIONAL,
  object-information [4] OCTET STRING OPTIONAL,
```

```
  on-start-up [5] ActionClass OPTIONAL,
  on-close-down [6] ActionClass OPTIONAL,
  original-group-cache-priority [7] INTEGER (0..255) DEFAULT 127,
  items [8] SEQUENCE SIZE (1..MAX) OF GroupItem OPTIONAL
}

StandardIdentifier ::= SEQUENCE
{
  joint-iso-itu INTEGER (2),
  mheg INTEGER (19)
}

GroupItem ::= CHOICE
{
  resident-program [9] ResidentProgramClass,
  remote-program [10] RemoteProgramClass,
  interchanged-program [11] InterchangedProgramClass,
  palette [12] PaletteClass,
  font [13] FontClass,
  cursor-shape [14] CursorShapeClass,
  boolean-variable [15] BooleanVariableClass,
  integer-variable [16] IntegerVariableClass,
  octet-string-variable [17] OctetStringVariableClass,
  object-ref-variable [18] ObjectRefVariableClass,
  content-ref-variable [19] ContentRefVariableClass,
  link [20] LinkClass,
  stream [21] StreamClass,
  bitmap [22] BitmapClass,
  line-art [23] LineArtClass,
  dynamic-line-art [24] DynamicLineArtClass,
  rectangle [25] RectangleClass,
  hotspot [26] HotspotClass,
  switch-button [27] SwitchButtonClass,
  push-button [28] PushButtonClass,
  text [29] TextClass,
  entry-field [30] EntryFieldClass,
  hyper-text [31] HyperTextClass,
  slider [32] SliderClass,
  token-group [33] TokenGroupClass,
  list-group [34] ListGroupClass
}

--A.3. Application Class _____
```

```
ApplicationClass ::= SET
{
  COMPONENTS OF GroupClass,
  on-spawn-close-down [35] ActionClass OPTIONAL,
  on-restart [36] ActionClass OPTIONAL,
  default-attributes [37] SEQUENCE SIZE (1..MAX) OF DefaultAttribute OPTIONAL
}

DefaultAttribute ::= CHOICE
{
  character-set [38] INTEGER,
  background-colour [39] Colour,
  text-content-hook [40] INTEGER,
  text-colour [41] Colour,
  font [42] FontBody,
  font-attributes [43] OCTET STRING,
  interchanged-program-content-hook [44] INTEGER,
  stream-content-hook [45] INTEGER,
  bitmap-content-hook [46] INTEGER,
  line-art-content-hook [47] INTEGER,
  button-ref-colour [48] Colour,
  highlight-ref-colour [49] Colour,
  slider-ref-colour [50] Colour
}

FontBody ::= CHOICE
{
  direct-font OCTET STRING,
  indirect-font ObjectReference
}

-- A.4 Scene Class ---------------------------------------------------------

SceneClass ::= SET
{
  COMPONENTS OF GroupClass,
  input-event-register [51] INTEGER,
  scene-coordinate-system [52] SceneCoordinateSystem,
  aspect-ratio [53] AspectRatio DEFAULT {width 4, height 3},
  moving-cursor [54] BOOLEAN DEFAULT FALSE,
  next-scenes [55] SEQUENCE SIZE (1..MAX) OF NextScene OPTIONAL
}

SceneCoordinateSystem ::= SEQUENCE
```

```
{
  x-scene INTEGER,
  y-scene INTEGER
}

AspectRatio ::= SEQUENCE
{
  width INTEGER,
  height INTEGER
}

NextScene ::= SEQUENCE
{
  scene-ref OCTET STRING,
  scene-weight INTEGER (0..255)
}
-- A.5. Ingredient Class ____________________________________________

 IngredientClass ::= SET
{
  RootClass (WITH COMPONENTS
    {..., external-reference (WITH COMPONENTS {..., object-number (1..MAX)})}),
  initially-active [56] BOOLEAN DEFAULT TRUE,
  content-hook [57] INTEGER OPTIONAL,
  original-content [58] ContentBody OPTIONAL,
  shared [59] BOOLEAN DEFAULT FALSE
}

ContentBody ::= CHOICE
{
  included-content OCTET STRING,
  referenced-content ReferencedContent
}

ReferencedContent ::= SEQUENCE
{
  content-reference ContentReference,
  content-size [60] INTEGER OPTIONAL,
  content-cache-priority [61] INTEGER (0..255) DEFAULT 127
}

-- A.6. Link Class _____________________________________________

LinkClass ::= SET
```

```
{
  COMPONENTS OF IngredientClass
    (WITH COMPONENTS {..., content-hook ABSENT, original-content ABSENT}),
  link-condition [62] LinkCondition,
  link-effect [63] ActionClass
}

LinkCondition ::= SEQUENCE
{
  event-source ObjectReference,
  event-type EventType,
  event-data EventData OPTIONAL
}

EventType ::= ENUMERATED
{
  is-available(1),
  content-available(2),
  is-deleted(3),
  is-running(4),
  is-stopped(5),
  user-input(6),
  anchor-fired(7),
  timer-fired(8),
  asynch-stopped(9),
  interaction-completed(10),
  token-moved-from(11),
  token-moved-to(12),
  stream-event(13),
  stream-playing(14),
  stream-stopped(15),
  counter-trigger(16),
  highlight-on(17),
  highlight-off(18),
  cursor-enter(19),
  cursor-leave(20),
  is-selected(21),
  is-deselected(22),
  test-event(23),
  first-item-presented(24),
  last-item-presented(25),
  head-items(26),
  tail-items(27),
  item-selected(28),
```

```
  item-deselected(29),
  entry-field-full(30),
  engine-event(31)
}

EventData ::= CHOICE
{
  octetstring OCTET STRING,
  boolean BOOLEAN,
  integer INTEGER
}

-- A.7 Program Class ____________________________________________

ProgramClass ::= SET
{
  COMPONENTS OF IngredientClass
    (WITH COMPONENTS {..., initially-active (FALSE) PRESENT}),
  name [64] OCTET STRING,
  initially-available [65] BOOLEAN DEFAULT TRUE
}

-- A.8 Resident Program Class ___________________________________

ResidentProgramClass ::= ProgramClass
  (WITH COMPONENTS {..., content-hook ABSENT, original-content ABSENT})

-- A.9 Remote Program Class _____________________________________

RemoteProgramClass ::= SET
{
  COMPONENTS OF ProgramClass
  (WITH COMPONENTS {..., content-hook ABSENT, original-content ABSENT}),
  program-connection-tag [66] INTEGER OPTIONAL
}

-- A.10 Interchanged Program Class ______________________________

InterchangedProgramClass ::= ProgramClass
  (WITH COMPONENTS {..., original-content PRESENT})

-- A.11 Palette Class ___________________________________________

PaletteClass ::= IngredientClass
```

```
  (WITH COMPONENTS
    {..., content-hook PRESENT, original-content PRESENT, initially-active (TRUE)})

-- A.12 Font Class ____________________________________________________

FontClass ::= IngredientClass
  (WITH COMPONENTS
    {..., content-hook PRESENT, original-content PRESENT, initially-active (TRUE)})

-- A.13 Cursor Shape ____________________________________________________

CursorShapeClass ::= IngredientClass
  (WITH COMPONENTS
    {..., content-hook PRESENT, original-content PRESENT, initially-active (TRUE)})

-- A.14 Variable Class ____________________________________________________

VariableClass ::= SET
{
  COMPONENTS OF IngredientClass
       (WITH COMPONENTS {..., content-hook ABSENT, original-content ABSENT, initially-active
(TRUE)}),
  original-value [67] OriginalValue
}

OriginalValue ::= CHOICE
{
  boolean BOOLEAN,
  integer INTEGER,
  octetstring OCTET STRING,
  object-reference [68] ObjectReference,
  content-reference [69] ContentReference
}

-- A.15 Boolean Variable Class ____________________________________

BooleanVariableClass ::= VariableClass
  (WITH COMPONENTS {..., original-value (WITH COMPONENTS {..., boolean PRESENT})})

-- A.16 Integer Variable Class ____________________________________

IntegerVariableClass ::= VariableClass
  (WITH COMPONENTS {..., original-value (WITH COMPONENTS {..., integer PRESENT})})
```

```
-- A.17 Octet String Variable Class _____

OctetStringVariableClass ::= VariableClass
  (WITH COMPONENTS {..., original-value (WITH COMPONENTS {..., octetstring PRESENT})})

-- A.18 Object Reference Variable Class _____

ObjectRefVariableClass ::= VariableClass
  (WITH COMPONENTS {..., original-value (WITH COMPONENTS {..., object-reference PRESENT})})

-- A.19 Content Reference Variable Class _____

ContentRefVariableClass ::= VariableClass
    (WITH COMPONENTS {..., original-value (WITH COMPONENTS {..., content-reference
PRESENT})})

-- A.20 Presentable Class _____

PresentableClass ::= IngredientClass

-- A.21 Token Manager Class _____

TokenManagerClass ::= SET
{
  movement-table [70] SEQUENCE SIZE (1..MAX) OF Movement OPTIONAL
}

Movement ::= SEQUENCE SIZE (1..MAX) OF INTEGER

-- A.22 Token Group Class _____

TokenGroupClass ::= SET
{
  COMPONENTS OF PresentableClass
    (WITH COMPONENTS {..., content-hook ABSENT, original-content ABSENT}),
  COMPONENTS OF TokenManagerClass,
  token-group-items [71] SEQUENCE SIZE (1..MAX) OF TokenGroupItem,
  no-token-action-slots [72] SEQUENCE SIZE (1..MAX) OF ActionSlot OPTIONAL
}

TokenGroupItem ::= SEQUENCE
{
  a-visible ObjectReference,
  action-slots SEQUENCE SIZE (1..MAX) OF ActionSlot OPTIONAL
```

```
}

ActionSlot ::= CHOICE
{
 action-class ActionClass,
 null NULL
}

-- A.23 List Group Class ---------------------------------------------

ListGroupClass ::= SET
{
 COMPONENTS OF TokenGroupClass,
 positions [73] SEQUENCE SIZE (1..MAX) OF XYPosition,
 wrap-around [74] BOOLEAN DEFAULT FALSE,
 multiple-selection [75] BOOLEAN DEFAULT FALSE
}

-- A.24 Visible Class ------------------------------------------------

VisibleClass ::= SET
{
 COMPONENTS OF PresentableClass,
 original-box-size [76] OriginalBoxSize,
 original-position [77] XYPosition DEFAULT {x-position 0, y-position 0},
 original-palette-ref [78] ObjectReference OPTIONAL
}

OriginalBoxSize ::= SEQUENCE
{
 x-length INTEGER (0..MAX),
 y-length INTEGER (0..MAX)
}

-- A.25 Bitmap Class -------------------------------------------------

BitmapClass ::= SET
{
 COMPONENTS OF VisibleClass
  (WITH COMPONENTS {..., original-content PRESENT}),
 tiling [79] BOOLEAN DEFAULT FALSE,
 original-transparency [80] INTEGER (0..100) DEFAULT 0
}
```

```
-- A.26 Line Art Class ____________________________________________

LineArtClass ::= SET
{
  COMPONENTS OF VisibleClass
    (WITH COMPONENTS {..., original-content PRESENT}),
  bordered-bounding-box [81] BOOLEAN DEFAULT TRUE,
  original-line-width [82] INTEGER DEFAULT 1,
  original-line-style [83] INTEGER {solid(1), dashed(2), dotted(3)} DEFAULT solid,
  original-ref-line-colour [84] Colour OPTIONAL,
  original-ref-fill-colour [85] Colour OPTIONAL
}

-- A.27 Rectangle Class ___________________________________________

RectangleClass ::= LineArtClass
  (WITH COMPONENTS
    {..., content-hook ABSENT, original-content ABSENT, bordered-bounding-box ABSENT})

-- A.28 Dynamic Line Art Class ____________________________________

DynamicLineArtClass ::= LineArtClass
  (WITH COMPONENTS
    {..., content-hook ABSENT, original-content ABSENT})

-- A.29 Text Class ________________________________________________

TextClass ::= SET
{
  COMPONENTS OF VisibleClass
    (WITH COMPONENTS {..., original-content PRESENT}),
  original-font [86] FontBody OPTIONAL,
  font-attributes [43] OCTET STRING OPTIONAL,
  text-colour [41] Colour OPTIONAL,
  background-colour [39] Colour OPTIONAL,
  character-set [38] INTEGER OPTIONAL,
  horizontal-justification [87] Justification DEFAULT start,
  vertical-justification [88] Justification DEFAULT start,
  line-orientation [89] LineOrientation DEFAULT horizontal,
  start-corner [90] StartCorner DEFAULT upper-left,
  text-wrapping [91] BOOLEAN DEFAULT FALSE
}

Justification ::= ENUMERATED
```

```
{
 start(1),
 end(2),
 centre(3),
 justified(4)
}

LineOrientation ::= ENUMERATED {vertical(1), horizontal(2)}

StartCorner ::= ENUMERATED
{
 upper-left(1),
 upper-right(2),
 lower-left(3),
 lower-right(4)
}

-- A.30 Stream Class ____________________________________________

StreamClass ::= SET
{
 COMPONENTS OF PresentableClass
  (WITH COMPONENTS {..., original-content PRESENT}),
 multiplex [92] SEQUENCE SIZE (1..MAX) OF StreamComponent,
 storage [93] Storage DEFAULT stream,
 looping [94] INTEGER {infinity(0)} DEFAULT 1
}

StreamComponent ::= CHOICE
{
 audio [95] AudioClass,
 video [96] VideoClass,
 rtgraphics [97] RTGraphicsClass
}

Storage ::= ENUMERATED {memory(1), stream(2)}

-- A.31 Audio Class _____________________________________________

AudioClass ::= SET
{
 COMPONENTS OF PresentableClass
  (WITH COMPONENTS {..., content-hook ABSENT, original-content ABSENT, shared ABSENT}),
 component-tag [98] INTEGER,
```

```
  original-volume [99] INTEGER DEFAULT 0
}

-- A.32 Video Class ____________________________________________________

VideoClass ::= SET
{
  COMPONENTS OF VisibleClass
     (WITH COMPONENTS {..., content-hook ABSENT, original-content ABSENT, shared ABSENT,
original-palette-ref ABSENT}),
  component-tag [98] INTEGER,
  termination [100] Termination DEFAULT disappear
}

Termination ::= ENUMERATED {freeze(1), disappear(2)}

-- A.33 RTGraphics Class _____________________________________________

RTGraphicsClass ::= SET
{
  COMPONENTS OF VisibleClass
    (WITH COMPONENTS {..., content-hook ABSENT, original-content ABSENT, shared ABSENT}),
  component-tag [98] INTEGER,
  termination [100] Termination DEFAULT disappear
}

-- A.34 Interactible Class ____________________________________________

InteractibleClass ::= SET
{
  engine-resp [101] BOOLEAN DEFAULT TRUE,
  highlight-ref-colour [49] Colour OPTIONAL
}

-- A.35 Slider Class _____________________________________________

SliderClass ::= SET
{
  COMPONENTS OF VisibleClass
    (WITH COMPONENTS {..., content-hook ABSENT, original-content ABSENT}),
  COMPONENTS OF InteractibleClass,
  orientation [102] Orientation,
  max-value [103] INTEGER,
  min-value [104] INTEGER DEFAULT 1,
```

```
  initial-value [105] INTEGER OPTIONAL,
  initial-portion [106] INTEGER OPTIONAL,
  step-size [107] INTEGER DEFAULT 1,
  slider-style [108] SliderStyle DEFAULT normal,
  slider-ref-colour [50] Colour OPTIONAL
}

Orientation ::= ENUMERATED {left(1), right(2), up(3), down(4)}

SliderStyle ::= ENUMERATED {normal(1), thermometer(2), proportional(3)}

-- A.36 Entry Field Class ---------------------------------------------

EntryFieldClass ::= SET
{
  COMPONENTS OF TextClass,
  COMPONENTS OF InteractibleClass,
  input-type [109] InputType DEFAULT any,
  char-list [110] OCTET STRING OPTIONAL,
  obscured-input [111] BOOLEAN DEFAULT FALSE,
  max-length [112] INTEGER DEFAULT 0
}

InputType ::= ENUMERATED {alpha(1), numeric(2), any(3), listed(4)}

-- A.37 Hyper Text Class ----------------------------------------------

HyperTextClass ::= SET
{
  COMPONENTS OF TextClass,
  COMPONENTS OF InteractibleClass
}

-- A.38 Button Class --------------------------------------------------

ButtonClass ::= SET
{
  COMPONENTS OF VisibleClass
    (WITH COMPONENTS {..., content-hook ABSENT, original-content ABSENT}),
  COMPONENTS OF InteractibleClass,
  button-ref-colour [48] Colour OPTIONAL
}

-- A.39 Hotspot Class -------------------------------------------------
```

```
HotspotClass ::= ButtonClass

-- A.40 Push Button Class ---------------------------------------------

PushButtonClass ::= SET
{
  COMPONENTS OF ButtonClass,
  original-label [113] OCTET STRING OPTIONAL,
  character-set [38] INTEGER OPTIONAL
}

-- A.41 Switch Button Class -------------------------------------------

SwitchButtonClass ::= SET
{
  COMPONENTS OF PushButtonClass,
  button-style [114] ButtonStyle
}

ButtonStyle ::= ENUMERATED
{
  pushbutton(1),
  radiobutton(2),
  checkbox(3)
}

-- A.42 Action Class --------------------------------------------------

ActionClass ::= SEQUENCE SIZE (1..MAX) OF ElementaryAction

ElementaryAction ::= CHOICE
{
  activate [115] GenericObjectReference,
  add [116] Add,
  add-item [117] AddItem,
  append [118] Append,
  bring-to-front [119] GenericObjectReference,
  call [120] Call,
  call-action-slot [121] CallActionSlot,
  clear [122] GenericObjectReference,
  clone [123] Clone,
  close-connection [124] CloseConnection,
  deactivate [125] GenericObjectReference,
```