EG 201 063 V1.1.2 (1997-08)

ETSI Guide

Multimedia Terminals and Applications (MTA); End-to-end protocols for multimedia information retrieval services; Application walkthrough in a DAVIC system

European Telecommunications Standards Institute

Reference
DEG/MTA-001069 (9uc00ide.PDF)

Keywords
MHEG, MMI, multimedia, terminal

ETSI Secretariat

Postal address
F-06921 Sophia Antipolis Cedex - FRANCE

Office address
650 Route des Lucioles - Sophia Antipolis
Valbonne - FRANCE
Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16
Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

X.400
c= fr; a=atlas; p=etsi; s=secretariat

Internet
secretariat@etsi.fr
http://www.etsi.fr

Contents

Intellectual Property Rights

ETSI has not been informed of the existence of any Intellectual Property Right (IPR) which could be, or could become essential to the present document. However, pursuant to the ETSI Interim IPR Policy, no investigation, including IPR searches, has been carried out. No guarantee can be given as to the existence of any IPRs which are, or may be, or may become, essential to the present document.

Foreword

This ETSI Guide (EG) has been produced by ETSI Technical Committee Terminal Equipment (TE) and approved by its successor ETSI Project Multimedia Terminals and Applications (MTA).

1 Scope

This ETSI Guide (EG) describes a walkthrough in a DAVIC system at the application level. The DAVIC specifications define a set of tools enabling the deployment of systems that support audio visual applications including TV distribution, near Video on Demand , Video on demand, teleshopping, etc.

At the application level, DAVIC has selected:

- the DSM-CC User to User protocol as the tool to be used for end to end communication between a client (a terminal equipment) and a server;
- the MHEG-5 final form representation as the tool to be used for application interchange.

On the basis of an end-user-oriented concrete scenario, this walkthrough aims at showing how these two technologies work together and complement themselves.

This walkthrough is illustrated with MHEG-5 objects (using the normative textual notation) and with pieces of code running at the terminal side (using a Java-like programming language).

2 References

References may be made to:

a) specific versions of publications (identified by date of publication, edition number, version number, etc.), in which case, subsequent revisions to the referenced document do not apply; or

b) all versions up to and including the identified version (identified by "up to and including" before the version identity); or

c) all versions subsequent to and including the identified version (identified by "onwards" following the version identity); or

d) publications without mention of a specific version, in which case the latest version applies.

A non-specific reference to an ETS shall also be taken to refer to later versions published as an EN with the same number.

[1] DAVIC specifications.

[2] ISO/IEC IS 13818-6 (1996): "Information technology - Generic coding of moving pictures and associated audio information - Part 6: Extension for Digital Storage Media Command and Control".

[3] ISO/IEC IS 13522-5 (1996): "Information technology - Coding of multimedia and hypermedia information - Part 5: Support for Base-Level Interactive Applications".

[4] ETS 300 777-2: "Terminal equipment (TE); End-to-end protocols for multimedia information retrieval services; Part 2: Use of for Digital Storage Media Command and Control (DSM-CC) for basic multimedia applications".

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the definitions in the DAVIC specifications [1], ISO/IEC IS 13818-6 [2] and ISO/IEC IS 13522-5 [3] apply.

3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

DAVIC	Digital Audio Visual Council
DSM-CC	Digital Storage Media Command and Control
IEC	International Electrotechnical Commission
IS	International Standard
ISO	International Organization for Standardization
MHEG	Multimedia and Hypermedia Expert Group
MTA	Multimedia Terminals and Applications
NSAP	Network Service Access Point
STU	Set Top Unit
U-U	User to User

4 The walkthrough scenario

The walkthrough scenario consists of ten steps:

- starting the service gateway welcome application;
- browsing the service gateway welcome application;
- starting the service provider welcome application;
- browsing the service provider welcome application;
- running the movies on demand application;
- playing a movie;
- stopping a movie;
- quitting the movies on demand application;
- quitting the service provider welcome application;
- quitting the service gateway welcome application.

4.1 Starting the service gateway welcome application

On the service gateway system, it is assumed that the DSM-CC tree is organized as described in the following example in which every element is given as a (`kind`, `id`) couple.

```
(DSM::ServiceGateway, aServiceGateway)
    (DSM::Directory, Services)
        (DSM::Directory, Welcome)          // welcome application
            (DSM::File, startup)
            (DSM::Directory, Scenes)
                (DSM::File, aScene)
                ...
            (DSM::Directory, Contents)
                (DSM::File, aContent)
                ...
            (DSM::Directory, InterchangedPrograms)
                (DSM::File, anInterchangedProgram)
                ...
            ...
        ...
    ...
```

When the user switches on his STU, the main resident application is automatically started. This application knows a local DSM session object reference it has to use to attach to the service gateway (for the purpose of the present document called `localSessionObjRef`). It knows as well the identification of the server hosting the service gateway, the logical name of the service gateway and the path to the first service on the service gateway. The STU main resident application shall invoke a DSM-CC U-U `Directory::attach` operation on the DSM session local object reference (`localSessionObjRef`) in order to get the object reference of the service gateway (`aServiceGatewayObjRef`) and the object reference of the first service (that is the welcome application) within the service gateway (`firstServiceObjRef`).

The `attach` operation prototype definition is given hereafter (see ISO/IEC IS 13818-6 [2] for more details).

```
void attach
    (
    in sequence<octet,20> serviceDomain,
    in CosNaming::Name pathName,
    in UserContext savedContext,
    out ObjRefs resolvedRefs
    )
    raises { ... };
```

Part of a possible code for the attachment to the service gateway is given hereafter using a Java-like programming language.

```
Session localSessionObjRef;
ServiceGateway aServiceGatewayObjRef;
Directory firstServiceObjRef;
Object[] resolvedRefs;
byte[] serviceDomain;
CosNaming::Name pathName;

// attach to the service gateway

// the server identification shall contain the globally unique server network
// address in NSAP address format
serviceDomain = ...;

// the path name shall contain the logical name of the service gateway
// followed by the path to the first service
pathName[1].kind = "DSM::ServiceGateway";
pathName[1].id = "aServiceGateway";
pathName[2].kind = "DSM::Directory";
pathName[2].id = "Services";
pathName[3].kind = "DSM::Directory";
pathName[3].id = "Welcome";

localSessionObjRef.attach
    (
    serviceDomain,
    pathName,
    null,               // no saved context
    resolvedRefs
    );

// the object reference of the service gateway will be returned in the first
// element of the array
aServiceGatewayObjRef = (ServiceGateway) resolvedRefs[1];

// the object reference of the first service within the service gateway
// (that is the welcome application) will be returned in the second element
// of the array
firstServiceObjRef = (Directory) resolvedRefs[2];
```

The STU main resident application shall then invoke a DSM-CC U-U `Directory::resolve` operation on the object reference of the service gateway welcome application in order to get the object reference of the application bootstrap.

The `resolve` operation prototype definition is given hereafter (see ISO/IEC IS 13818-6 [2] for more details).

```
Object resolve(in Name n) raises { ... };
```

The service gateway welcome application is MHEG-5 encoded and therefore contains by definition a unique `Application` object to be used as a bootstrap. This object is stored in a DSM-CC file called by convention "startup". The `resolve` operation is invoked with this name and returns the object reference of the DSM-CC file.

Part of a possible code for the `resolve` operation invocation is given hereafter using a Java-like programming language.

```
Directory firstServiceObjRef;
File applicationObjRef;
CosNaming::Name name;

name[1].kind = "DSM::File";
name[1].id = "startup";
applicationObjRef = (File) firstServiceObjRef.resolve(name);
```

The STU local application shall then download the `Application` object in invoking the DSM-CC U-U `File::read` operation on the object identified by the `applicationObjRef` object reference.

The `read` operation prototype definition is given hereafter (see ISO/IEC IS 13818-6 [2] for more details).

```
void read
    (
    in u_longlong aOffset,
    in u_longlong aSize,
    in boolean aReliable,
    out opaque rData
    )
    raises { ... };
```

Part of a possible code for the `read` operation invocation is given hereafter using a Java-like programming language.

```
File applicationObjRef;
byte[] application;

applicationObjRef.read
    (
    0L,                                             // from the beginning
    applicationObjRef.getContentSize(),             // the whole content
    true,                                           // reliable RPC
    application
    );
```

Finally, the STU local application shall launch the MHEG-5 engine on the `Application` object.

4.2 Browsing the service gateway welcome application

From a functional point of view, the service gateway welcome application is in charge of presenting the available services to the user. It allows for example:

- to easily browse the services directory (e.g. through the use of a scroll list);
- to get more information about a service by:
  a) consulting a standardized description; or
  b) playing a specific overview (designed by the service provider);
- and finally, to connect to a service.

These welcome applications usually use navigation metaphors like the mall metaphor. They often propose commercial animation such as dynamic ads, games, etc.

From a technical point of view, the service gateway welcome application is an MHEG-5 encoded application. It is made of a single `Application` object (called "startup") and several `Scene` objects. `Application` and `Scene` objects are the interchanged units. They are composed of `Ingredient` objects used to present information to the user, to interact with him and to express the behaviour of the application.

It is assumed that from the standardized description of a service the user can either go back to the services directory or connect to the service. The first choice will launch a new scene within the current application, the second one will launch a new application.

An example of part of a `Scene` object (called "aServiceDescription" in the "aServiceGateway/Services/Welcome/Scenes/aServiceDescription" directory) representing the standardized description of a given service is given hereafter using the MHEG-5 textual notation.

```
{:Scene

// MHEG object reference: object identifier + object number
(~/Scenes/aServiceDescription 0)

:Items                                  // list of items
(
```

The `Bitmap` object numbered 1 represents the bitmap to be used as a background. It references a content (called "background" in the "aServiceGateway/Services/Welcome/Contents" directory). As it is by default initially active, this object is activated when its embedding scene (called "aServiceDescription" in the "aServiceGateway/Services/Welcome/Scenes" directory) is activated. Its activation initiates:

- the downloading of the referenced content through the invocation of the DSM-CC U-U `Directory::resolve` and `File::read` operations;
- the displaying of the bitmap on the screen.

```
// bitmap to be used as a background
{:Bitmap
  1                                     // object number
  :OrigContent                          // bitmap content
    :ContentRef                         // content reference
      "~/Contents/background"
  :OrigBoxSize                          // bitmap size (full screen)
    640                                 // length along X-axis
    480                                 // length along Y-axis
  :OrigPosition                         // bitmap position
    0                                   // X position
    0                                   // Y position
}
```

Part of a possible code for the referenced content downloading is given hereafter using a Java-like programming language.

```
Directory firstServiceObjRef;
File bitmapObjRef;
byte[] bitmap;
CosNaming::Name name;

name[1].kind = "DSM::Directory";
name[1].id = "Contents";
name[2].kind = "DSM::File";
name[2].id = "background";
bitmapObjRef = (File) firstServiceObjRef.resolve(name);

bitmapObjRef.read
    (
    0L,                                         // from the beginning
    bitmapObjRef.getContentSize(),              // the whole content
    true,                                       // reliable RPC
    bitmap
    );
```

The `PushButton` object numbered 2 represents the push button which allows the user to go back to the services directory. It is associated with a `Link` object (numbered 3) which is fired when the push button is selected, and triggers a transition to the scene representing the services directory. The `TransitionTo` elementary action initiates:

- the deactivation and the deletion of the current scene (called "aServiceDescription" in the "aServiceGateway/Services/Welcome/Scenes" directory);
- the preparation of the new scene (called "servicesDirectory" in the same directory) which includes its downloading through the invocation of the DSM-CC U-U `Directory::resolve` and `File::read` operations;
- the activation of the new scene.

```
// push button to go back to the services directory
{:PushButton
  2                                   // object number
  :OrigBoxSize                        // push button size
    160                               // length along X-axis
    20                                // length along Y-axis
  :OrigPosition                       // push button position
    0                                 // X position
    0                                 // Y position
  :OrigLabel                          // push button label
    "Back to the services directory"
}

// link to trigger the transition to the services directory
{:Link
  3                                   // object number
  :EventSource                        // object number of the associated push
    2                                 // button
  :EventType
    IsSelected
  :LinkEffect
    (
      :TransitionTo
        (
          (                                 // object
            "~/Scenes/servicesDirectory"    // reference of
            0                               // the targeted
          )                                 // scene
        )
    )
}
```

Part of a possible code for the new scene downloading is given hereafter using a Java-like programming language.

```
Directory firstServiceObjRef;
File sceneObjRef;
byte[] scene;
CosNaming::Name name;

name[1].kind = "DSM::Directory";
name[1].id = "Scenes";
name[2].kind = "DSM::File";
name[2].id = "servicesDirectory";
sceneObjRef = (File) firstServiceObjRef.resolve(name);

sceneObjRef.read
    (
    0L,                                         // from the beginning
    sceneObjRef.getContentSize(),               // the whole content
    true,                                       // reliable RPC
    scene
    );
```

4.3 Starting the service provider welcome application

The `PushButton` object numbered 4 represents the push button which allows the user to connect to the service. It is associated with a `Link` object (numbered 5) which is fired when the push button is selected and triggers the spawning of a new application. This is done through the use of the `Call` elementary action followed by a `Spawn` elementary action.

```
// push button to connect to the service
{:PushButton
  4                                   // object number
  :OrigBoxSize                        // push button size
    160                               // length along X-axis
    20                                // length along Y-axis
  :OrigPosition                       // push button position
    160                               // X position
    0                                 // Y position
  :OrigLabel                          // push button label
    "Connect to the service"
}

// link to trigger the connection to the service
{:Link
  5                                   // object number
  :EventSource                        // object number of the associated push
    4                                 // button
  :EventType
    IsSelected
  :LinkEffect
    (
      :Call
        (
          6                           // object number of the interchanged program
          7                           // object number of a boolean variable
                                      // to store the completion status of the
                                      // call
          :GOctetString               // first parameter : server
            "..."                     // identification
          :GOctetString               // second parameter : service gateway
            "aServiceProvider"        // logical name
          :GOctetString               // third parameter : path to the first
            "Services/Welcome"        // service
        )
      :Spawn
        (                             // object reference
          ("~/startup" 0)             // of the targeted
        )                             // application
    )
}
```

The `Call` elementary action targets an `InterchangedProgram` object (numbered 6) in charge of transferring to a new service (detach from the current service gateway and to attach to the new one). This interchanged program uses the DSM-CC U-U `Session::detach` and `Session::attach` operations.

```
{:InterchgPrg
  6                                   // object number
  :OrigContent                        // interchanged program content
    :ContentRef                       // content reference
      "~/Programs/ServiceTransfer"
  :Name                               // interchanged program name
    "ServiceTransfer"
}

{:BooleanVar
  7                                   // object number
  :OriginalValue
    false
}

) // end of list of items
}
```

The `detach` operation prototype definition is given hereafter (see ISO/IEC IS 13818-6 [2] for more details).

```
void detach
    (
    in boolean aSuspend,
    out UserContext savedContext
    )
    raises { ... };
```

Part of a possible code included in (or referenced by) the `InterchangedProgram` object is given hereafter using a Java-like programming language.

```
Session localSessionObjRef;
ServiceGateway aServiceGatewayObjRef;
Directory firstServiceObjRef;
Object[] resolvedRefs;
byte[] serviceDomain;
CosNaming::Name pathName;

// detach from the current service gateway
aServiceGatewayObjRef.detach(false,null);

// attach to the new one

// the server identification shall contain the globally unique server network
// address in NSAP address format
// it shall be set to the value of the interchanged program first parameter
serviceDomain = ...;

// the path name shall contain the logical name of the service gateway
// followed by the path to the first service
pathName[1].kind = "DSM::ServiceGateway";
// the following identifier shall be set to the value of the interchanged
// program second parameter
pathName[1].id = ...;
pathName[2].kind = "DSM::Directory";
// the following identifier shall be extracted from the value of the
// interchanged program third parameter
pathName[2].id = ...;
pathName[3].kind = "DSM::Directory";
// the following identifier shall be extracted from the value of the
// interchanged program third parameter
pathName[3].id = ...;

localSessionObjRef.attach
    (
    serviceDomain,
    pathName,
    null,               // no saved context
    resolvedRefs
    );

// the object reference of the new service gateway will be returned in the
// first element of the array
aServiceGatewayObjRef = (ServiceGateway) resolvedRefs[1];

// the object reference of the first service within the new service gateway
// (that is the welcome application) will be returned in the second element
// of the array
firstServiceObjRef = (File) resolvedRefs[2];
```

NOTE: The `detach` operation is invoked on the remote service gateway object reference; the `attach` operation is invoked on the local DSM session object reference.

The `Spawn` elementary action initiates:

- the deactivation and the deletion of the current `Scene` object (called "aServiceDescription" " in the "aServiceGateway/Services/Welcome/Scenes");
- the deactivation and the deletion of the current `Application` object (called "startup" in the "aServiceGateway/Services/Welcome" directory);

- the preparation of the new `Application` object (called "startup" in the "aServiceProvider/Services/Welcome" directory) to be launched which includes its downloading through the invocation of the DSM-CC U-U `Directory::resolve` and `File::read` operations;

- the activation of the new `Application` object.

4.4 Browsing the service provider welcome application

The aim of the service provider welcome application is to guide the user through the bouquet of services.

On the service provider system, it is assumed that the DSM-CC tree is organized as described in the following example in which every element is given as a (`kind`, `id`) couple.

```
(DSM::ServiceGateway, aServiceProvider)
    (DSM::Directory, Services)
        (DSM::Directory, Welcome)           // welcome application
            (DSM::File, startup)
            (DSM::Directory, Scenes)
                (DSM::File, aScene)
                ...
            (DSM::Directory, Contents)
                (DSM::File, aContent)
                ...
            ...
        (DSM::Directory, MoviesOnDemand)    // movies on demand application
            (DSM::File, startup)
            (DSM::Directory, Scenes)
                (DSM::File, aScene)
                ...
            (DSM::Directory, Contents)
                (DSM::File, aContent)
                ...
            (DSM::Directory, RemotePrograms)
                (DSM::File, aRemoteProgram)
                ...
            (DSM::Directory, Movies)
                (DSM::Stream, aMovie)
                ...
            ...
        ...
    ...
```

To browse the service provider welcome application, the same techniques and mechanisms are used as for the service gateway welcome application: a transition from a scene to another scene initiates the downloading of the targeted scene, the presentation of a referenced content initiates the downloading of the content and so on.

When the user chooses one of the proposed services, the corresponding application is spawned. In our scenario, we assume that the user chooses the movies on demand service.

4.5 Running the movies on demand application

The movies on demand application guides the user through the catalogue of available movies. A detailed description is associated with each movie which gives all the relevant information (title, date, duration, director's name, casting, language, subtitled or not, etc.) including the cost. The description is illustrated with the film poster. The consultation of a trailer is offered as well. From that description, the user can play the movie.

An example of part of a `Scene` object (called "aMovieDescription" in the "aServiceProvider/Services/MoviesOnDemand/Scenes" directory) representing the detailed description of a given movie is given hereafter using the MHEG-5 textual notation.

```
{:Scene

// MHEG object reference: object identifier + object number
(~/Scenes/aMovieDescription 0)

:Items                              // list of items
(
```

The PushButton object numbered 1 represents the push button which allows the user to play the movie.

```
// push button to select the movie
{:PushButton
  1                                 // object number
  :OrigBoxSize                      // push button size
    160                             // length along X-axis
    20                              // length along Y-axis
  :OrigPosition                     // push button position
    0                               // X position
    0                               // Y position
  :OrigLabel                        // push button label
    "Play the movie"
}
```

It is associated with a Link object (numbered 2) which is fired when the push button is selected and triggers the presentation of a dialogue for the user to enter the password. This is done through the use of the Run elementary action targeted at the objects representing this dialogue.

```
// link to trigger the password dialogue
{:Link
  2                                 // object number
  :EventSource
    1                               // object number of the associated push button
  :EventType
    IsSelected
  :LinkEffect
    (
      :Run
        (3)                         // object number of the targeted text
      :Run
        (4)                         // object number of the targeted entry field
    )
}
```

The Text object numbered 3 represents the text to ask the user for his password.

The EntryField object numbered 4 represents the entry field for the user to enter its password. The entered characters shall not be echoed in a readable form (the ObscuredInput exchanged attribute is set to true).

These two objects are not initially activated with their embedding scene (the InitiallyActive exchanged attribute is set to false) but shall be explicitly activated through the use of the Run elementary action.

```
// text to ask for the password
{:Text
  3                                   // object number
  :InitiallyActive                    // the text is NOT initially active
    false
  :OrigContent                        // text content
    "Enter your password"
  :OrigBoxSize                        // bounding box size
    160                               // length along X-axis
    20                                // length along Y-axis
  :OrigPosition                       // bounding box position
    240                               // X position
    220                               // Y position
}

// entry field to receive the password
{:EntryField
  4                                   // object number
  :InitiallyActive                    // the entry field is NOT initially active
    false
  :OrigBoxSize                        // bounding box size
    160                               // length along X-axis
    20                                // length along Y-axis
  :OrigPosition                       // bounding box position
    240                               // X position
    240                               // Y position
  :ObscuredInput                      // echo shall not be readable
    true
}
```

The entry field is associated with a `Link` object (numbered 5) which is fired at the end of the interaction and triggers:

- the copy of the interaction result into a variable (through the use of the `GetTextData` elementary action);
- with this variable as an input parameter, the call of a remote program in charge of authenticating the entered password (trough the use of the `Call` elementary action);
- the generation of an event if the authentication result is successful (through the use of the `TestVariable` elementary action) in order to fire a link.

```
// link to trigger the authentication remote program call
{:Link
  5                                   // object number
  :EventSource                        // object number of the associated entry
    4                                 // field
  :EventType
    InteractionCompleted
  :LinkEffect
    (
      :GetTextData
        (
          4                           // object number of the targeted entry field
          6                           // object number of an octet string variable to store the
                                      // entry field content
        )
      :Call
        (
          7                           // object number of the targeted remote program
          8                           // object number of a boolean variable to store the
completion status of the call
          :GOctetString               // first parameter : password to
            :IndirectRef              // authenticate
              6
          :Gboolean                   // second parameter : authentication
            :IndirectRef              // result
              9
        )
      :TestVariable
        (
          7                           // object number of the targeted
                                      // variable
          1                           // operator : 1 means equal
          :Gboolean                   // comparison value
            :true
        )
    )
}

// octet string variable to store the entry field content
{:OStringVar
  6                                   // object number
}


{:RemotePrg
  7                                   // object number
  :Name                               // remote program name
    "~/RemotePrograms/Authenticate"
}


// boolean variable to store the call completion status
{:BooleanVar
  8                                   // object number
  :OriginalValue
    false
}

// boolean variable to store the authentication result
{:BooleanVar
  9                                   // object number
  :OriginalValue
    false
}
```

The DAVIC 1.1 specifications define in part 5 a mapping for the MHEG-5 `call` and `fork` elementary actions targeted at remote programs. This mapping is as follows:

```
module DAV
{
 enum parType {boolPar, intPar, octStringPar, objRefPar, contRefPar};
 typedef sequence<octet> octString;
 typedef struct oR {
  octString groupIdentifier;
  long objectNumber;
 } objRef;
 typedef octString contRef;
 union par switch (parType)
 {
  case boolPar: boolean aBoolean;
  case intPar: long anInt;
  case octStringPar: octString aString;
  case objRefPar: objRef anObjRef;
  case contRefPar: contRef aContRef;
 };
 typedef sequence<par> pars;
 interface MHEG {
  void call(
   in octString procedureName,
   inout pars somePars
  );
  void fork(
   in octString procedureName,
   inout pars somePars
  );
 };
};
```

This mapping allows the MHEG-5 engine at the terminal side to call a remote program at the server side and to exchange parameters with this program.

Part of a possible code to call the authentication remote program is given hereafter using a Java-like programming language.

```
MHEG MHEGObjRef;
Object[] somePars;

// set the procedure name to the Name attribute of the RemoteProgram object
String procedureName = new String(...);

// set the first parameter to the Value attribute of the OctetStringVariable object
somePars[1] = (Object) new String(...);

MHEGObjRef.call(procedureName, somePars);

// set the Value attribute of the BooleanVariable object to the second
// parameter
... = ((Boolean) somePars[2]).booleanValue();
```

NOTE 1: In this example it is assumed that the object reference allowing the calling of remote programs (`MHEGObjRef`) is returned to the client together with the service gateway object reference (`serviceGatewayObjRef`) and the first service object reference (`firstServiceObjRef`).

NOTE 2: At the server side, DAVIC does not define the communication mechanisms between the server and the program to be called: is the program a procedure, a thread, a process ? how are the parameters exchanged: through a pipe, a socket, a shared memory area, etc. ? Hence this is implementation dependant.

The `Link` object (numbered 10) is associated to the variable returned by the remote program. It is fired when the value of the variable is true and triggers the playing of the movie.

```
// link to trigger the playing of the movie
{:Link
  10                                    // object number
  :EventSource
    9                                   // object number of the associated
                                        // variable
  :EventType
    TestEvent
  :EventData
    true
  :LinkEffect
    (
      :TransitionTo
        (
          "~Scenes/aMovie"              // object reference of
          0                             // the targeted scene
        )
    )
}
```

NOTE 3: In this example neither the objects to represent the treatment of an unsuccessful remote program call nor the objects to represent the treatment of an unsuccessful authentication result are described.

4.6 Playing a movie

Once the user has chosen a movie, this movie is played. During the presentation he can use a VCR-like control panel which allows him to:

- pause the movie;
- resume the movie;
- play the movie faster in the forward direction;
- play the movie faster in the reverse direction;
- increase the volume;
- decrease the volume.

An example of part of a `Scene` object (called "aMovie" in the "aServiceProvider/Services/MoviesOnDemand/Scenes" directory) representing the movie is given hereafter using the MHEG-5 textual notation.

```
{:Scene

// MHEG object reference: object identifier + object number
(~/Scenes/aMovie 0)

:Items                              // list of items
(
```

The `Stream` object numbered 1 represents the stream which contains the movie. As it is by default initially active, this object is activated when its embedding scene is activated. Its activation initiates:

- the resolution of the stream name through the invocation of the DSM-CC U-U `Directory::resolve` operation;
- the starting of the stream presentation through the invocation of the DSM-CC U-U `Stream::resume` operation.

```
// stream which contains the movie
{:Stream
  1                                 // object number
  :OrigContent                      // stream content
    :ContentRef                     // content reference
      "~/Streams/aMovie"
  :OrigBoxSize                      // presentation box original size
    640                             // length along X-axis
    460                             // length along Y-axis
  :OrigPosition                     // presentation box original position
    0                               // X position
    0                               // Y position
  :Multiplex
    (
      {:Video
        2                           // object number
        :ComponentTag
          1
      }
      {:Audio
        3                           // object number
        :ComponentTag
          2
      }
    )
}
```

Part of a possible code for the resolution of the stream name is given hereafter using a Java-like programming language.

```
Directory MoDServiceObjRef;
Stream aMovieObjRef;
CosNaming::Name name;

name[1].kind = "DSM::Directory";
name[1].id = "Streams";
name[2].kind = "DSM::Stream";
name[2].id = "aMovie";
aMovieObjRef = (Stream) MoDServiceObjRef.resolve(name);
```

The `resume` operation prototype definition is given hereafter (see ISO/IEC IS 13818-6 [2] for more details).

```
void resume
    (
    in AppNPT rStart,
    in Scale rScale
    )
    raises { ... };
```

Part of a possible code for starting the stream presentation is given hereafter using a Java-like programming language.

```
Stream aMovieObjRef;

// play now
AppNPT rStart = new AppNPT(0x80000000, 0);

// play forward at the normal viewing rate
Scale rScale = new Scale(1,1);

aMovieObjRef.resume(rStart,rScale);
```

The `PushButton` object numbered 4 represents the push button which allows the user to pause the movie.

```
// push button to pause the movie
{:PushButton
  4                                 // object number
  :OrigBoxSize                      // push button size
    80                              // length along X-axis
    20                              // length along Y-axis
  :OrigPosition                     // push button position
    0                               // X position
    460                             // Y position
  :OrigLabel                        // push button label
    "Pause the movie"
}
```

It is associated to a `Link` object (numbered 5) which is fired when the push button is selected and triggers the pausing of the movie. This is done through the use of the `SetSpeed` elementary action targeted at the `Video` object representing the video stream within the `Stream` object representing the movie.

```
// link to trigger the pausing of the movie
{:Link
  5                                     // object number
  :EventSource
    4                                   // object number of the associated push button
  :EventType
    IsSelected
  :LinkEffect
    (
      :SetSpeed
        (
          2                             // object number of the targeted video
          0                             // new speed numerator
          1                             // new speed denominator (could be
                                        // omitted as 1 is the default value)
        )
    )
}
```

The `SetSpeed` elementary action initiates the pausing of the stream presentation through the invocation of the DSM-CC U-U `Stream::pause` operation.

The `pause` operation prototype definition is given hereafter (see ISO/IEC IS 13818-6 [2] for more details).

```
void pause
    (
    in AppNPT rStop
    )
    raises { ... };
```

Part of a possible code for pausing the stream presentation is given hereafter using a Java-like programming language.

```
Stream aMovieObjRef;

// pause now
AppNPT rStop = new AppNPT(0x80000000, 0);

aMovieObjRef.pause(rStop);
```

Another `PushButton` object is defined to allow the user to resume the movie. It is associated with a `Link` object embedding a `SetSpeed` elementary action with a `NewSpeed` parameter value of 1/1. This action initiates the invocation of the DSM-CC U-U `Stream::resume` operation with a `rStart` parameter value meaning "now" and a `rScale` parameter value meaning "play forward at the normal viewing rate".

Another PushButton object is defined to allow the user to play the movie faster in the forward direction. It is associated with a Link object embedding a SetSpeed elementary action with a NewSpeed parameter value of n/1 (it is up to the application designer to choose the value of n). This action initiates the invocation of the DSM-CC U-U Stream::resume operation with an rStart parameter value meaning "now" and an rScale parameter value meaning "play forward at n times the normal viewing rate".

Another PushButton object is defined to allow the user to play the movie faster in the reverse direction. It is associated with a Link object embedding a SetSpeed elementary action with a NewSpeed parameter value of -n/1. This action initiates the invocation of the DSM-CC U-U Stream::resume operation with an rStart parameter value meaning "now" and an rScale parameter value meaning "play reverse at n times the normal viewing rate".

The PushButton object numbered 6 represents the push button which allows the user to increase the volume.

```
// push button to pause the movie
{:PushButton
  6                                 // object number
  :OrigBoxSize                      // push button size
    80                              // length along X-axis
    20                              // length along Y-axis
  :OrigPosition                     // push button position
    320                             // X position
    460                             // Y position
  :OrigLabel                        // push button label
    "Increase the volume"
}
```

It is associated with a Link object (numbered 7) which is fired when the push button is selected and triggers the volume increasing. This is done through the use of:

- the GetVolume elementary action targeted at the Audio object representing the audio stream within the Stream object representing the movie to set a Variable object with the current volume;
- the Add elementary action targeted at this Variable object to increase its value by one;
- the SetVolume elementary action targeted at the Audio object to set the volume with the updated value of the Variable object.

```
// link to trigger the volume increasing
{:Link
  7                                 // object number
  :EventSource
    6                               // object number of the associated push button
  :EventType
    IsSelected
  :LinkEffect
    (
      :GetVolume
        (
          3                         // object number of the targeted audio
          8                         // object number of the variable to
                                    // store the volume
        )
      :Add
        (
          8                         // object number of the targeted
                                    // variable
          1                         // value to add
        )
      :SetVolume
        (
          3                         // object number of the targeted audio
          :IndirectRef
          8                         // object number of the variable to
                                    // retrieve the volume
        )
    )
}

// integer variable to store the volume
{:IntegerVar
  8                                 // object number
  :OriginalValue
    0
}
```

Another `PushButton` object is defined to allow the user to decrease the volume. It is associated with a `Link` object embedding a `GetVolume`, a `Subtract` and a `SetVolume` elementary actions.

4.7 Stopping a movie

At any time during the presentation the user can choose to stop it and go back to the previous scene (that is the movie description).

A `PushButton` object is defined for that purpose. It is associated to a `Link` object embedding a `TransitionTo` elementary action targeted at the scene representing the movie description.

This transition to a new scene initiates the deactivation and the deletion of the current scene.

The deactivation of the scene triggers the deactivation of all active embedded ingredients.

In the present case, the deactivation of the stream will stop the presentation through the invocation of the DSM-CC U-U `Stream::pause` operation.

The deletion of the scene triggers the deletion of all active embedded ingredients.

In the present case, the deletion of the stream will free all resources allocated to that stream and the deletion of the scene itself will free all the resources allocated to that scene. Amongst these resources the object reference of the DSM-CC stream and the object reference of the DSM-CC file will be freed through the invocation of the DSM-CC U-U `Base::close` operation.

NOTE: If the MHEG-5 engine implements a caching strategy, it may decide when deleting an object either to actually free all associated resources or to cache them. Using the `GroupCachePriority` exchanged attribute of the object, the application designer can allow or disallow the caching.

The `close` operation prototype definition is given hereafter (see ISO/IEC IS 13818-6 [2] for more details).

```
void close();
```

Part of a possible code for freeing the object reference of the DSM-CC stream and the object reference of the DSM-CC file is given hereafter using a Java-like programming language.

```
Stream aMovieObjRef;
File aSceneObjRef;

aMovieObjRef.close();
aSceneObjRef.close();
```

4.8 Quitting the movies on demand application

Within the movies on demand application there shall be a means for the user to quit the application and restart the service provider welcome application.

A `PushButton` object is defined for that purpose (at the main menu level for example). It is associated with a `Link` object embedding a `Quit` elementary action targeted at the `Application` object of the movies on demand application. This will initiate the deletion of the current scene, the deletion of the currently active `Application` object and the restarting of the last application that has been pushed in the stack - that is, the service provider welcome application.

NOTE: Application stacking is an optional feature of the MHEG-5 engine. If the engine does not provide the functionality, the `Launch` elementary action (corresponding to the `goto` instruction of a programming language) shall be used instead of the `Spawn` and `Quit` elementary actions (corresponding to the `call` and `return` instructions of a programming language).

4.9 Quitting the service provider welcome application

Within the service provider welcome application there shall be a means for the user to quit the application and restart the service gateway welcome application.

A `PushButton` object is defined for that purpose (at the main menu level for example). It is associated with a `Link` object embedding a `Call` elementary action followed by a `Quit` elementary action targeted at the `Application` object of the service provider welcome application.

The `Call` elementary action targets a `ResidentProgram` object (numbered 1) in charge of detaching from the current service gateway and attaching back to the initial service gateway. This resident program uses the DSM-CC U-U `Session::detach` and `Session::attach` operations.

```
{:ResidentPrg
  1                                 // object number
  :Name                             // resident program name
    "BackToTheServiceGateway"
}
```

4.10 Quitting the service gateway welcome application

Within the service gateway welcome application there shall be a means for the user to quit the application and shutdown the terminal.

A `PushButton` object is defined for that purpose (at the main menu level for example). It is associated with a `Link` object embedding `Quit` elementary action targeted at the `Application` object of the service gateway welcome application.

As there are no more applications pushed in the stack, the MHEG engine shall stop when executing this `Quit` elementary action and give control back to the STU main resident application.

The STU main resident application shall then shutdown - that is, free all the resources (and especially network resources) which includes detaching from the service gateway.

The STU is then ready to be switched off.

5 Conclusions

This application walkthrough shows that the subset of the DAVIC specifications relevant to the present document is implementable, and fulfils the application requirements.

History

Document history			
V1.1.1	May 1997	Membership Approval Procedure	MV 9730: 1997-05-27 to 1997-07-25
V1.1.2	August 1997	Publication	

ISBN 2-7437-1644-4
Dépôt légal : Août 1997