FD – Media Content Manager

Content

History

Version	Date	Adjustments
A	2006-10-02	First revision. (MMAWI)

1 Introduction

The purpose of this document is to explain the internal structure and functions of the Media Content Manager component. See ref. [1] for specifications of the Media Content Manager functions.

1.1 Overview

The Media Content Manager provides access to Media Content Packages. A Media Content Package contains Media Content. An application may use a number of Media Content Packages to for example support multiple languages.

The functionality of the Media Content Manager is divided into two parts:

1. A separate application - The Media Content Installer, which handles the installation of Media Content Package's.
2. An MASP runtime component – Media Content Manager Component, which provides runtime access to the installed Media Content Packages.

Note: To distinctly separate the two parts in this document, the separate application will be called **Media Content Installer** and the platform component **Media Content Manager**.

The two "parts" are described in the following two sections.

1.2 Introduction to Media Content Installer

The Media Content Installer, which is a shell script, manages the installation/uninstallation of Media Content Packages. A Media Content Package (MCP) is a deliverable that contains a set of media files together with their configuration.

The install/uninstall functionality is lifted to a separate stand-alone application as it must be possible to administrate packages without running the entire MASP.

1.3 Introduction to Media Content Manager

The Media Content Manager manages the lookup and retrieval of Media Content Resources, i.e. installed Media Content Packages.

2 Media Content Installer

2.1 Function Structure

The Media Content Installer consists of a shell script, mcpadmin.sh, which provides the following functionality:

- Install Media Content Packages
- Uninstall Media Content Packages
- List all installed Media Content Packages

2.2 Function Behavior

2.2.1 Install Media Content Package

The *install* option installs an MCP to the Media Content Manager.

The script, mcpadmin.sh, takes as input an MCP and installs it. MCPs are deliverables created by the standalone application ApplicationAndMediaContentPackageManager, (see [2] for details).

The *mediaContentPackage* is a file reference of a bundled set of configuration files that together with the media files comprise the entire Media Content Package.

The MediaContentPackage has the following structure:

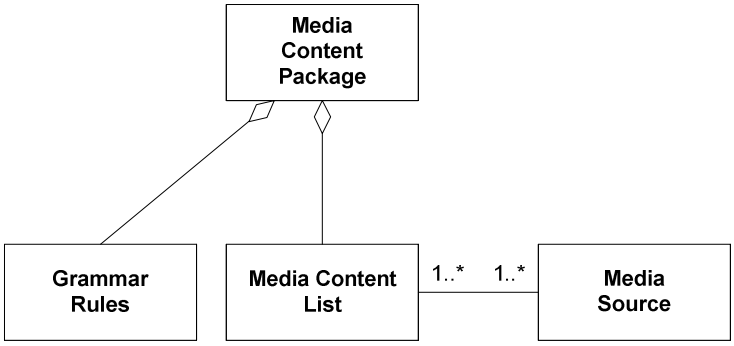

Figure 1 Media Content Package

The installation consists of writing the package in the format and location that the Media Content Manager expects and to register the package in MCR.

2.2.2 Uninstall Media Content Package

The *uninstall* option uninstalls an MCP from the Media Content Manager. The MCP to uninstall is given by argument to the script.

The uninstallation removes all files associated with the MCP and unregisters the package in MCR.

2.2.3 List installed Media Content Packages

The *view* option lists information about all MCPs currently installed.

3 Media Content Manager

3.1 Overview

The Media Content Manager provides an interface to installed Media Content Packages.

Figure 2 shows the modules that comprise the Media Content Manager. The arrows denote dependencies. As seen in Figure 2, Media Content Manager Component is dependent on the Media Object Component.

The content manager creates a resource for each content package that is installed into the application. The resource uses the following:

- MediaObject component for creating Media Objects.
- MediaObjectCache for caching created MediaObjects
- ConditionInterpreter to evaluate what message of many in a content that is to be played.
- GrammarRules

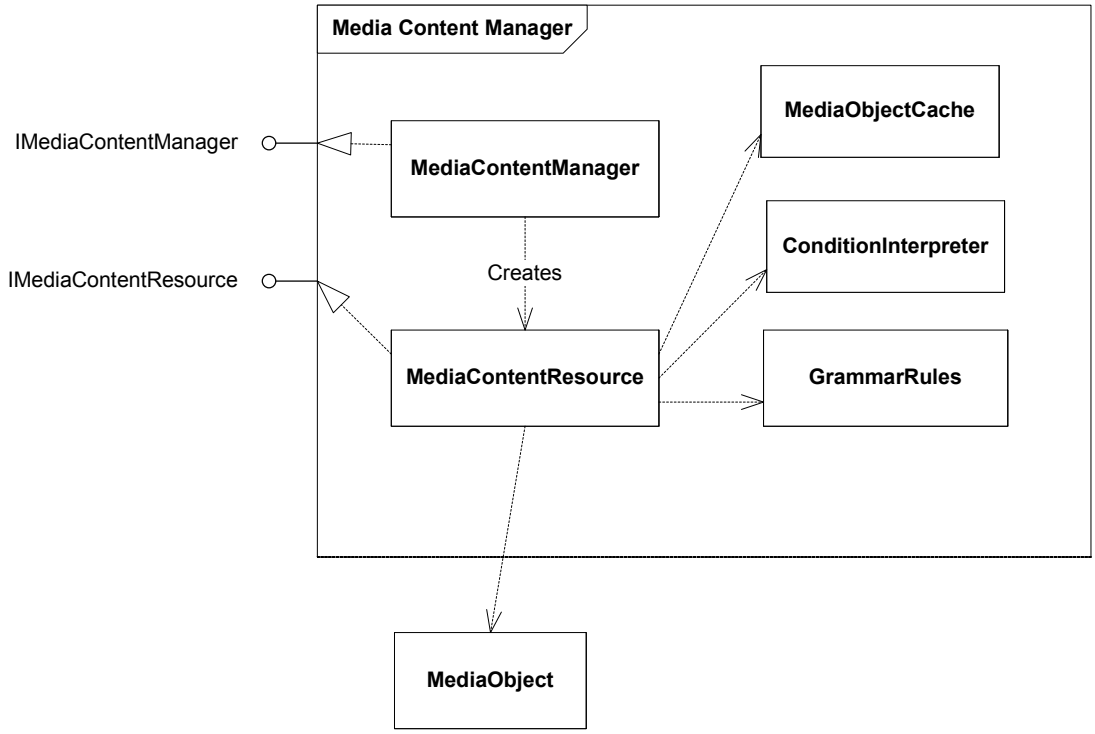

Figure 2 Modules in Media Content Manager

3.2 MediaContentManager

3.2.1 Function Structure

The MediaContentManager provides the top-level interface to Media Content Packages, via the IMediaContentManager interface. The interface provides methods to lookup media content with certain properties and retrieve it.

On startup the MediaContentManager reads the Media Content Packages from the file system. The structure of the package is described in the following subsection.

3.2.1.1 Media Content Package File Structure

A Media Content Package is a collection of files which is interpreted by the MediaContentManager on startup. The top-level file is the Media Content Package

Definition File. Figure 3 shows an overview of the file structure of Media Content Packages.

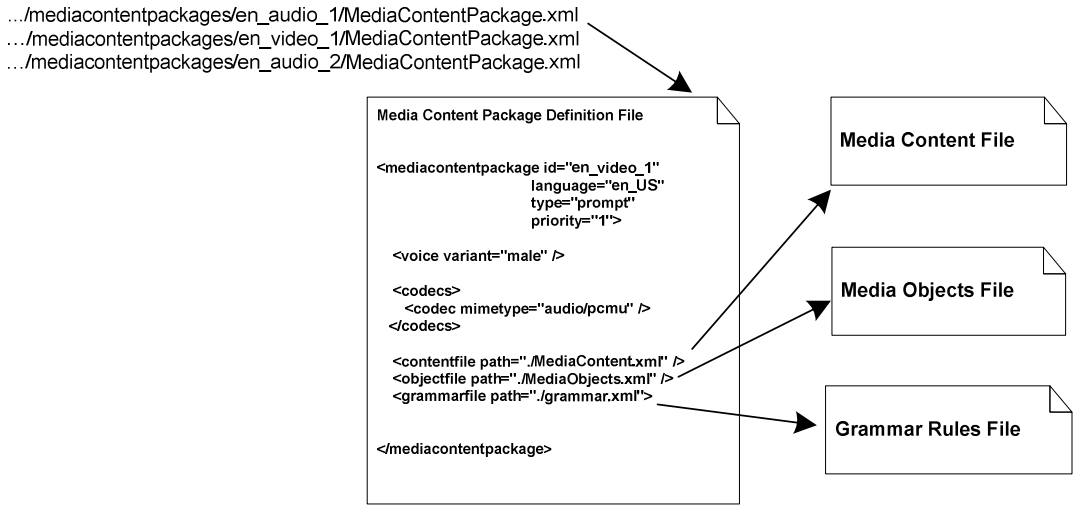

Figure 3 File structure of a Media Content Package

3.2.1.1.1 Media Content Package Definition File

The Media Content Package Definition File is the top-level file and contains the basic properties of the contained media and references to all other files in the package. The properties given are:

- Id of the package
- Language
- Type
- priority
- Video Variant
- Voice Variant
- Codecs

The referenced files are:

- Media Content File
- Media Objects File
- Grammar Rule File

A Media Content Package may have several Media Content files and Media Object files.

3.2.1.1.2 Media Content File

The *`Media Contents`* files list all contents of a package. Figure 4 shows a snapshot of a media contents file.

Media Contents File

```
<mediacontents>

  <mediacontent id="1">
    <qualifiers/>
    <instance cond="true" />
      <element type="mediafile" reference="VVA_0001.mov" />
    </instance>
  </mediacontent>

  <mediacontent id="2">
    <qualifiers>
      <qualifier name="numberOfDigits" type="Number" gender="None" />
    </qualifiers>
    <instance cond="true" />
      <element type="mediafile" reference="VVA_0008.mov" />
      <element type="qualifier" reference="numberOfDigits:Number:None" />
    </instance>
  </mediacontent>

  <mediacontent id="3">
    <qualifiers>
      <qualifier name="numberOfNewMessages" type="Number" gender="None" />
    </qualifiers>
    <instance cond="((numberOfNewMessages == 1))" />
      <element type="mediafile" reference="VVA_0276.mov" />
    </instance>
    <instance cond="((numberOfNewMessages > 1))" />
      <element type="qualifer" reference="numberOfNewMessages:Number:none" />
      <element type="mediafile" reference="VVA_0277.mov" />
    </instance>
  </mediacontent>

  <mediacontent id="subject" returnall="true">
    <qualifiers>
      <qualifier name="forward" type="String" gender="None"/>
      <qualifier name="urgent" type="String" gender="None"/>
      <qualifier name="confidential" type="String" gender="None"/>
      <qualifier name="messageType" type="String" gender="None"/>
      <qualifier name="sender" type="MediaObject" gender="None"/>
    </qualifiers>
    <instance cond="(forward == 'true')">
      <element type="text" reference="forward"/>
    </instance>
    <instance cond="(urgent == 'true')">
      <element type="text" reference="urgent"/>
    </instance>
    <instance cond="(confidential == 'true')">
      <element type="text" reference="confidential"/>
    </instance>
    <instance cond="(messageType == 'voice')">
      <element type="text" reference="voice"/>
    </instance>
    <instance cond="(messageType == 'video')">
      <element type="text" reference="video"/>
    </instance>
    <instance cond="true">
      <element type="text" reference="from"/>
    </instance>
    <instance cond="true">
      <element type="qualifier" reference="sender:MediaObject:None"/>
    </instance>
  </mediacontent>
</mediacontents>
```

Figure 4 Media Contents File

Each *content* has a unique *id* which is referenced from the application and one or many *message instances*. Each *message instance* has a *condition* which is interpreted at runtime by the *Media Content Manager* in order to select the one that meets the input. The *qualifiers* is the input to this interpretation, but can also be part of the message (converted and played). Each message instance consists of one or many message elements which is either a reference to a mediafile or a qualifier that is converted to its media object representation. If the type is mediafile or text the reference will point to a media object in the Media Objects file where the actual content resides (see Media Objects file in next section).

A *content* may have the attribute *returnall*. If this is set to *true*, all *message instances* which *condition* is interpreted as true will be selected. In this way a message consisting of several message elements may be built up condition by condition. An instance for each possible combination of conditions is not needed. See the example with subject in Figure 4 and in table below.

Table 1 shows the logical representation of the content file in Figure 4.

Table 1 Media Contents representation

Media Content Id	Condition	Media Source Reference
1	true	VVA_0001.mov
2	true	VVA_0008.mov, numberOfDigits:Number:None
3	numberOfNewMessages == 1	VVA_0276.mov
3	numberOfNewMessages > 1	numberOfNewMessages:Number:None, VVA_0277.mov
subject	forward=='true'	forward
	urgent=='true'	urgent
	confidential=='true'	confidential
	messageType=='voice'	voice
	messageType=='video'	video
	true	from
	true	sender:MediaObject:None

Explanation of content with id 3:

The content with id 3 above has two messages, each with a condition. The Media Content Manager will at run-time interpret each condition with the qualifier named *numberOfNewMessages* as input. If *numberOfNewMessages=1* the first message is selected which will result in file VVA_0276.mov being played. If *numberOfNewMessages=2* the second message is selected which will result in the qualifier itself being converted and played, i.e. "two", followed by the file VVA_0277.mov.

Explanation of content with id subject:

The content with id 'subject' above has the attribute returnall = true. It will be combined from several messages; all that has a condition interpreted as true will be appended.

First, the condition with the qualifier forward is interpreted. If forward is 'true', then the text message element "forward" is appended. The message element is a reference to a text media object in the Media Objects file, the actual text for "forward" is defined there.

Then the next condition, urgent, is interpreted. If this is satisfied, the text message element "urgent" is appended.

In the same way, dependent of the value of the qualifier messageType, one of the text message elements containing the text for message type is selected.

Then the text message element "from" and at last the sender. The result may then for example be "Forwarded urgent voice message from Kalle", dependent of how the text is defined in the Media Object file.

See *3.2.1.2 Condition language syntax* for a description of the condition language.

3.2.1.1.3 Media Objects File

The Media Objects Files list all media files used in the package, and static text used by the application such as "forward" used to build a mail. Figure 5 shows a snapshot of a Media Objects file.

MediaObjects.xml

```
<mediaobjects>

   <mediaobject type="MediaFile" src="VVA_0001.mov">
      <sourcetext><![CDATA[Good morning.]]></sourcetext>
   </mediaobject>

   <mediaobject type="Text"  src="forward">
      <sourcetext><![CDATA[Forwarded ]]></sourcetext>
   </mediaobject>

   <mediaobject type="Text"  src="voicemessage">
      <sourcetext><![CDATA[Voice message ]]></sourcetext>
   </mediaobject>

</mediaobjects>
```

Figure 5 Media Objects File

Each mediaobject listed has a *type* and a *src* attribute. The *src* attribute is significant as this is the label which identifies the media object from the *Media Content File*. The *type* attribute is either MediaFile or Text. If *MediaFile* the *src* attribute will be a reference to a media file, if *Text* the *src* attribute is just a label.

The *sourcetext* tag contains the text of the media. If the media is of type *Text* this is the actual content.

Each mediaobject has also one or many lengths. If many they have different units.

3.2.1.1.4 Grammar Rules File

The grammar rules file, see Figure 6, is used by the MediaContentManager to decide how numbers, dates and time shall be played in spoken words. For example, "21" shall in English be played as "twenty" "one", but in German "ein" "und" "zwanzig".

Grammar Rules File

```
<grammar>
  <rule type="Number" gender="None">
    <condition divisor="20"
               atomic="false"
               quotientFrom="1"
               quotientTo="1"
               remainderFrom="0"
               remainderTo="9"
               terminal="true"
               divide="true">

      <action type="mediafile">20</action>
    </condition>
    ...
    <condition divisor="1"
               atomic="true"
               quotientFrom="1"
               quotientTo="1"
               remainderFrom="0"
               remainderTo="0"
               terminal="true"
               divide="true">

      <action type="mediafile">1</action>
    </condition>
  </rule>
</grammar>
```

Figure 6 Grammar Rules File

The idea of the interpretation is to apply a sequence of divisions to the input number according to the defined division rules in the grammar rules file. As the number is divided, a list of mediaobjects is built up on the basis of the result of each division.

The grammar rule file is categorized in *rule* sections for each type of number and gender. A *rule* can have only one *type*, but many *gender*s. In each rule, the conditions for the divisions are defined in condition elements. Each condition has a *divisor* and a value range for the result of the division (the quotient and remainder). If the quotient and the remainder after the division fall into the range defined by the condition, the condition is satisfied. The conditions *action*s shall then be applied. The *condition* elements have the following attributes:

divisor	The input number will be divided by this number.
atomic	A condition is atomic if there is no need to further process the result after the division. For example, in English, a condition that tests for 7 should be atomic, 7 is only decomposed as "seven". 23, however, is not atomic. It is composed from 20 and 3: "twenty three". If an atomic condition is satisfied, the number decomposition ends with that result.
quotientFrom	The quotient range's lower bound. That is, the lowest value a quotient can have for this condition to be satisfied.
quotientTo	The quotient range's upper bound. That is, the highest value a quotient can have for this condition to be satisfied.

remainderFrom	The remainder range's lower bound. That is, the lowest value a remainder can have for this condition to be satisfied.
remainderTo	The remainder range's upper bound. That is, the highest value a remainder can have for this condition to be satisfied.
terminal	If a condition is not terminal, the quotient will be checked further before this condition's actions are applied. For example in English rule, if 200 is decomposed, we will get a quotient 2 when it is divided by 100. This quotient should also be decomposed to see how many "hundreds" there are. The result is then "two" "hundred".
divide	If the condition is divide, the checked number should be divided by the condition's divisor. If the condition is then satisfied, no more conditions are tested for this divisor. If the condition is not divide, the next condition will also be tested even if this condition is satisfied.

The conditions are grouped by the divisors and the condition tests are performed group by group, in descending order. Once a condition is satisfied within a divisor's group, no more conditions in that group are tested. If a condition implies that the number should be decomposed further, the conditions in the next divisor group is tested.

When the number is divided down to 0, the decomposition is finished. The result is now a list of actions. The actions can have the following types:

mediafile	A mediafile action has a reference to a mediaobject. The referenced file name is not complete, the file extension is added by MCM using the ContentTypeMapper to map the media content package's codec to a file extension.
swap	A swap action has a swap value, a positive or negative integer. This action determines that the next mediafile in the list will swap position with the one at the position specified by the swap value. E.g. if the swap value is -2, the next mediafile will change position with the mediafile two positions to the left.
skip	A skip action means that the next mediafile should only be added to the resulting list if it will not be the first in the list. For example can this be used to prevent a result to start with "and". If the result of some decomposition of time should be "one hour and one minute", the result would be "and one minute" if there is no hour. If

	skip is used for the "and", it will not be added in the beginning.
select	A select action can be used to select between different mediafiles dependent of their position in the list. A select action is placed between two mediafile actions. The second of the selectable mediafiles is used if this is the last in the result list. If other mediafiles will be added to the end of the resulting list, the first of the selectable mediafiles is used instead.

The list of actions is post processed and each mediafile action's reference is added to the resulting list of mediaobjects. The position of each mediaobject is determined by the other actions.

After the post processing, the result is a list of mediaobjects in the correct order that represents the input number.

Example:

Consider the input number 21. Figure 6 shows two conditions used in this example, but a complete grammar rules file has many more conditions.

The input number is divided by 20. The result is quotient=1 and remainder=1. Both the quotient and remainder falls into the range of the first condition above, so that condition is satisfied. The condition is *terminal*, so we will not process the quotient further. The condition is *divide*, so no other condition with divisor 20 will be tested. The condition is not *atomic*, so we will proceed with the remainder. This condition's action is added to the list of actions.

We now continue with the number 1. The second condition in the example above will be satisfied. This condition is atomic, so we are now done with the decomposition. The condition's action is added to the list of actions.

The action list now has two actions. Both actions are of the type mediafile, so the post processor returns "20" and "1". The file extension is then fetched from the ContentTypeMapper with the current Media Content Resource's codec, e.g. "audio/pcmu". The resulting file extension is ".wav". This results in the mediaobjects "20.wav" and "1.wav".

3.2.1.2 Condition language syntax

The message conditions are described in a custom, proprietary language. The language is typed, meaning that it supports type safety in some sense. The supported data types and their syntax are listed in Table 2.

Table 2 Qualifier data types

Data type	Syntax	Description	Example
Number	n	An integer.	1
DateDM	DateDM('yyyy-mm-dd')	A date in the form yyyy-mm-dd.	DateDM('2000-01-01')

CompleteDate	CompleteDate('yyyy-mm-dd hh:mm:ss +UTC')	A complete date with timezone in the form yyyy-mm-dd HH:MM:SS +UTC.	CompleteDate('2000-01-01 00:00:01 +0100')
WeekDay	WeekDay('yyyy-mm-dd')	The weekday of the date yyyy-mm-dd.	WeekDay('2000-01-01')
Time12	Time12('HH:MM:SS')	12 hour representation of a 24 hour time.	Time12('24:13:14')
Time24	Time24('HH:MM:SS')	24 hour time.	Time24('24:13:14')
String	'<the string>'	A string of characters.	'Hello MAS'

All types, except for String and Number, are explicitly denoted with its type signature.

In comparisons and arithmetics, types may not be mixed. For example, comparing a number and a date is illegal.

Each of the types uses different decomposition rules as described in 3.2.2.3.1.

The supported arithmetic operators for each qualifier types are shown in Table 3.

Table 3 Arithmetic operators for qualifiers

Data type	Operators supported	Description	Example
Number	+, -	Subtraction and addition.	1+1
DateDM, CompleteDate, WeekDay, Time12, Time24	-	Subtraction. Will return the difference in milliseconds.	DateDM('2000-01-01') – DateDM('2000-01-02')
String	None		

The supported relational operators for each qualifier type are shown in Table 4.

Table 4 Ralational operators for qualifiers

Data type	Operators supported	Description	Example
Number	==, !=, <, >, <=, >=	Equal, not equal, less than, greater than, less than or equal and greater than or equal.	1000 >= 2000
DateDM, CompleteDate, WeekDay, Time12, Time24	<, >	Less than, greater than. (i.e. before, after.)	DateDM('2000-01-01') < DateDM('2000-01-02')
String	==, !=	Equals, not equals	'Kalle' != 'Anka'

The supported operational operators are shown in Table 5.

Table 5 Operational operators

Operator	Description	Example
&&	AND	op1 && op2
||	OR	op1 || op2

3.2.1.2.1 Examples of legal conditions

`Condition: date > DateDM('2000-01-01')`

`Where:` date is a variable of type DateDM

True if variable date is greater than the constant date DateDM('2000-01-01'), i.e. a date later in time that 1st of January 2000.

`Condition: name == 'John Doe'`

`Where:` name is a variable of type String.

True if the value of variable name is 'John Doe'.

3.2.1.2.2 Examples of illegal conditions

`1 > DateDM('2000-01-01')`

Illegal as the type of the left hand side is a number and the right hand side is of type DateDM.

3.2.1.3 Media Content Manager Memory Structure

The language packages installed is read at startup by the MediaContentManager. The memory structure follows very closely the structure of the files, see 3.2.1.1-Media Content Package File Structure. Figure 7 shows the memory representation of a language package.

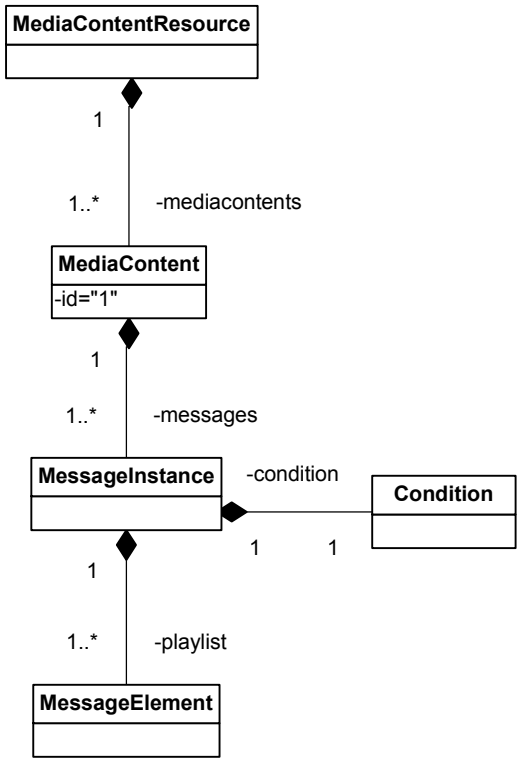

Figure 7 Memory structure of a Media Content Resource

A resource has one or many contents, each associated with a unique id. A content has one or many messages, which each has a condition. And finally a message has one or many elements, which forms the playlist of that particular message.

3.2.2 Function Behavior

3.2.2.1 Startup

On startup the MediaContentManager reads the installed Media Content Packages from the file system and builds an in-memory structure of the information needed to do fast lookup and retrieval. The actual media files are typically not read at startup.

3.2.2.2 Retrieval of Media Content Resources

The retrieval of media content is done with the method:

IMediaContentResource[] getMediaContentResource(IMediaFilter filter)

This method returns a list of resources that fulfills the filter. The filter contains a selection of the criterias for the following Media Content Resource characteristics:

- Language – As described by the Accept-Language header field in [RFC2068], example:
  - sv, en-gb;q=0.8, en;q=0.7
- Type – For example "Prompt", "Fun Greeting" etc

- Variant – Voice variant, for example "Female", "Male", and video variant, for example, "blue".
- Codecs – List of encodings in the content. For example ["audio/pcmu", "video/h263"]

The order in which the resources are returned is significant i.e. the first in the list has highest priority. The client may then select which resource(s) to use and retrieve Media Content from.

3.2.2.3 Retrieval of Media Content

Media Content is provided by a Media Content Resource. As described previously each resource will have a memory representation of the content in package files on the file system. It will NOT however load any media files into memory until asked for. Media Content is requested by specifying its identity and possibly a number of qualifiers (for example a date or number). The resource returns a list of Media Objects, represented by the IMediaObject interface, to the client.

If none of the conditions for a content is evaluated as true, an empty list of Media Object is returned.

If a content with an id that is not defined in the Media Content file is requested, an IllegalArgumentException is thrown.

Example:

Revisit the example in section 3.2.1.1.2 - Media Content File, where the content with id 3 was returned. The resource uses the in memory structure of the package to lookup the content with id 3, for each message in the content interpret its condition and return the message which condition is fulfilled, represented by an array of IMediaObjects.

If *numberOfNewMessage=1*, the file *VVA_0276.mov* is converted to an *IMediaObject* and returned.

If *numberOfNewMessage=2*, the *numberOfNewMessage* qualifier is converted to its *IMediaObject* representation, as described in 3.2.1.1.4, the file *VVA_0276.mov* is converted to an *IMediaObject* and the two media objects is then returned as an array with the qualifiers as first object.

3.2.2.3.1 Conversion of Qualifiers

The Media Content Manager supports conversion of qualifier values to MediaObjects for the following qualifier types:

Number	The value is decomposed using the number grammar rules described in 3.2.1.1.4.
DateDM	The value is decomposed using the number grammar rules described in 3.2.1.1.4.
Time12	The value is decomposed using the number grammar rules described in 3.2.1.1.4.
Time24	The value is decomposed using the number grammar rules described in 3.2.1.1.4.

WeekDay	The value of the qualifier is converted to a name of a day, using three letters, e.g. "mon". A MediaObject representing the day of week is returned, e.g. "mon.wav". The file extension is determined by the Media Content Resource's codecs.
String	A list of MediaObjects is returned, one MediaObject for each of the characters in the string, e.g. "a.wav", "b.wav". The file extension is determined by the Media Content Resource's codecs. Only characters a-z,A-Z,0-9 are handled, all others are discarded.
MediaObject	No conversion is performed, the input MediaObject is returned.

3.2.2.4 Caching of IMediaObjects

If the Media Content Manager is given a Media Object Cache, created IMediaObjects will be cached. The specific size and policy of the cache is configurable.

The 3PP ShiftOne Java Object Cache (see [3]) is used as implementation of this Media Object cache

4 References

[1] FS – Media Content Manager
16/0363-PRO049 Uen

[2] FS – ApplicationAndMediaContentPackageManager
31/0363-PRO049 Uen

[3] ShiftOne Java Object Cache
http://jocache.sourceforge.net/

5 Terminology

MCP	Media Content Package
MCR	Messaging Component Register