Customization Guide MAS

Operation Directions

ERICSSON

Contents

1 Introduction

1.1 Scope

This document provides information on how the MAS component can be customized.

1.2 Audience

The document is primarily written for personnel responsible for development and customization.

1.2.1 Prerequisite Knowledge

The readers of this document should have a good understanding of Messaging-over-IP and good knowledge and experience of:

- UNIX

1.3 Related Documents

The following documents also contain information related to the component, or are referred to from this document:

- Installation Guide MAS
- Operation & Maintenance MAS
- Solaris Documentation (http://docs.sun.com) *System Administration Guide, Solaris 10*

1.4 Document Conventions

1.4.1 Notational Conventions

This document uses the following notational conventions:

Bold font style is used for emphasis, to indicate keywords and buttons.

Italic font style is used for references, window/page/menu titles and specific terms.

`Bold Monospace font` is used to describe user input.

`Monospace font` is used for code, paths and on-screen computer output.

Square brackets ‘‘[]’’ are used to enclose parameters that are optional.

Curly brackets ‘‘{}’’ are used to enclose parameter values given as examples.

Less than and greater than characters ‘‘<>’’ are used to enclose variable names.

Backslash ‘‘\’’ at the end of a line means that this line is continued onto the next line.

The dollar sign ‘‘$’’ is the UNIX Korn (and Bourne) shell prompt.

The percent sign ‘‘%’’ is the UNIX C shell prompt.

The number sign ‘‘#’’ is the superuser prompt.

The number sign ‘‘#’’ is also used for comments.

A star ‘‘*’’ matches zero or more characters.

A question mark ‘‘?’’ matches one character.

A tilde ‘‘~’’ means the home directory of the current user.

‘‘<MAS_HOME>’’ is the installation directory for MAS, for example ‘‘/apps/mas’’.

2 General

The MAS Application and Media Content Package building software is a development kit for building Media Content Packages and Application Packages for the MAS component. The development kit is delivered in a container file described in Section 2.1 Container File on page 3. This kit is used to produce additional language that can be installed on an MAS component .

First, take a look at Section 2.1 Container File on page 3 , to find out what the container file contains.

Then, go through all steps in Section 3 Building an Media Content Package on page 5 , to produce the Media Content Package or Section 4 Building an Application Package on page 11 to produce an Application Package.

2.1 Container File

The MAS language building software is delivered in a container file named:

- `amcpm.<rel>.swu0044.1.solaris10`

where "<rel>" gives the release number of the software.

In this document, the container file is called **`<amcpm_builder_file>`**

The container file contains a directory hierarchy as follows:

amcpm/	Root directory including all directories and files described below.
amcpm/mkapp.sh	The script to use to build an Application Package
amcpm/mkmcp.sh	The script to use to build a Media Content Package.
amcpm/app_properties.cfg	The configuration file that specifies the characteristics of the Application Package to be built. This file shall be modified to reflect the actual application characteristics.
amcpm/mcp_properties.cfg	The configuration file that specifies the characteristics of the Media Content Package to be built. This file shall be modified to reflect the actual media characteristics.
amcpm/VERSION	Shows the version of the amcpm_builder_file.
amcpm/tools/grammarfiletool/*	Grammar File Tool. Used to check the number rules in a grammar rule file.

2.2 Pre-requisites

The MAS Application and Media Content Package builder requires that the following software has been installed on the host where the software is built:

- Solaris 10 or later with the latest recommended patches

2.3 Prepare building environment

1. Login to the building host.
2. Copy the MAS Application and Media Content Package delivery file to a any directory (here called <building_dir>) on the host.

   ```
   # cp <amcpm_builder_file> <building_dir>
   ```

3. Change to the building directory.

   ```
   # cd <building_dir>
   ```

4. Extract files from the MAS Application and Media Content Package delivery file.

   ```
   # tar xf <amcpm_builder_file>
   ```

3 Building an Media Content Package

3.1 General

This chapter describes how to build a Media Content Package that can be installed on a MAS component.

A Media Content Package has the following characteristics:

Type	An arbitrarly human readable tag that indicates the semantics of the Media Content Package, for example "prompt", "fungreeting" etc.
Language	Language as specified in the RFC 2798
Voice Variant	An arbitrarly human readable tag that indicates the voice variant, for example, "bob", "male" etc.
Video Variant	An arbitrarly human readable tag that indicates the video variant, for example, "blue", "autumn" etc.
Audio encoding	The encoding used for the audio in this package. For example "audio/pcmu", "audio/amr" etc.
Video encoding	The encoding used for the video in this package. For example "video/h263" etc

A Media Content Package can be built using any combination of the listed properties above.

3.2 Building Workflow

The building of a Media Content Package includes the steps described in subsections below. These steps should be performed in the following order:

a Section 3.3 Configure mcp_properties.cfg on page 5

b Section 3.4 Build the delivery file on page 9

3.3 Configure mcp_properties.cfg

The mcp_properties.cfg file defines the characteristics of the Media Content Package to be built.

1. Change to the <building_dir>/amcpm directory

```
# cd <building_dir>/amcpm
```

2. Edit the mcp_properties.cfg. See Section 3.3.1 Explanation of the mcp_properties.cfg file on page 6 for details.
3. Save the file and exit the editor.

3.3.1 Explanation of the mcp_properties.cfg file

The mcp_properties.cfg contains a number of properties possible to modify according to the description below.

Explanation:

mediafiles

The name (including path) to the mediafiles to be included for this Media Content Package. The path can be specified using wildcards or by listing the files one by one, e.g

- mediafiles=/vobs/ipms/vva/prompts/*.wav
- mediafiles=/vobs/ipms/vva/prompts/myfile.wav /vobs/ipms/vva/prompts/yourfile.wav

This property is mandatory.

mediacontentfiles

The name (including path) to the mediacontentfiles to be included for this Media Content Package. The path can be specified using wildcards or by listing the files one by one, e.g

- mediacontentfiles=/vobs/ipms/vva/mc/*.xml
- mediacontentfiles=/vobs/ipms/vva/mc/MediaContent_Voice.xml /vobs/ipms/vva/mc/MediaContent_Voice_System.xml

A media content file lists all Media Content Id:s as specified in #sce. This property is mandatory.

mediaobjectfiles

The name (including path) to the mediaobjectfiles to be included for this Media Content Package. The path can be specified using wildcards or by listing the files one by one, e.g

- mediaobjectfiles=/vobs/ipms/vva/mo/*.xml
- mediaobjecttfiles=/vobs/ipms/vva/mo/MediaObject_Voice.xml /vobs/ipms/vva/mo/MediaObject_System.xml

A media object file lists all Media Objects as specified in #sce. This property is mandatory.

grammarfile

The name (including path) to the grammarfile to be included for this Media Content Package. There can be only one grammar file specified e.g.

- grammarfile=/vobs/ipms/vva/prompts/grammar.xml

For details about the grammar file see Section 5 Appendix, Writing Grammar Rule files on page 17. This property is optional.

name

The name of the Media Content Package. This part is used to identify the Media Content Package as a customer adaptation

- name=ca_vva

This property is mandatory.

type

The type of the Media Content Package and is the same as the type described in Section 3.1 General on page 5 e.g.

- type=prompt

This property is mandatory.

customer

The name of the customer the Media Content Package is intended for. This part can include both the customer name as well as the CIS number.

- customer=customername.cis0001

This property is mandatory.

productid

The product identifier for the Media Content Package. Together with the variant and type this will be used to uniquely identify a Media Content Package e.g.

- productid=mcp0001/1

This property is mandatory.

rstate

The R-state of the Media Content Package e.g.

- rstate=p2a_007

lang

The language of the Media Content Package and is the same as the language described in Section 3.1 General on page 5 e.g.

- lang=en

This property is mandatory.

voicevariant The voice variant of the Media Content Package and is the same as the voice variant described in Section 3.1 General on page 5 e.g.

- voicevariant=female

One of voicevariant or videovariant must be specified.

videovariant The video variant of the Media Content Package and is the same as the video variant described in Section 3.1 General on page 5 e.g.

- videovariant=flowers

One of voicevariant or videovariant must be specified.

audioencoding The encoding of the audiopart of the Media Content Package and is the same as the audioencoding described in Section 3.1 General on page 5 e.g.

- audioencoding=audio/pcmu

This property is mandatory.

videoendcoding The encoding of the video part of the Media Content Package and is the same as the the videoencoding described in Section 3.1 General on page 5.

- videoencoding=video/quicktime

This property is mandatory if videovariant is used.

3.3.2 Example on mcp_properties.cfg

```
mediafiles=/vobs/ipms/vva/prompts/*.wav
mediacontentfiles=
 /vobs/ipms/vva/mediacontents/MediaContent_Voice_English.xml
mediaobjectfiles=
 /vobs/ipms/vva/mediaobjects/MediaObjects_Voice_English.xml
grammarfile=/vobs/ipms/vva/grammar/grammar.xml
name=ca_vva
type=prompt
customer=company.cis0001
productid=mcp0001
rstate=r1a.001
lang=sv
videovariant=video1
audioencoding=audio/pcmu
videoencoding=video/h263
```

The deliveryfilename is obtained from the properties file as follows:

<name>.<type>.<lang>.<videovariant>.<customer>.<rstate>.<productid>.tar

or

<name>.<type>.<lang>.<voicevariant>.<customer>.<rstate>.<productid>.tar

for example:

```
ca_vva.prompt.sv.video1.company.cis0001.r1a.001.mcp0001.tar
```

3.4 Build the delivery file

a Goto the location where the build script is located:

cd <building_dir>

b Build the delivery file by issuing the following command (if the <mcppropertiescfg> argument is left out in the command line the script tries to use mcp_properties.cfg file in the current directory if i exits):

./mkmcp.sh [<mcppropertiescfg>]

c A printout on the terminal shows where the delivery file can be obtained.

3.5 Example

Below is shown an example of the printouts when building a Media Content Package.

```
#./mkmcp.sh
Gather Media Content Package properties ...
About to create ca_vva.prompt.sv.video1.company.\
cis0001.r1a.001.mcp0001 Media
Content Package!
Copy files to temporary directory ...
Make delivery file ...
./mediatmp/
./mediatmp/media/
./mediatmp/media/0.wav
./mediatmp/media/1.wav
./mediatmp/media/10.wav
./mediatmp/media/100.wav
./mediatmp/media/11.wav
./mediatmp/media/VVA_0001.wav
./mediatmp/media/VVA_0002.wav
./mediatmp/media/VVA_0003.wav
./mediatmp/mediacfg/
./mediatmp/mediacfg/MediaContent.xml
./mediatmp/mediacfg/MediaObjects.xml
./mediatmp/mediacfg/MediaObjects_System.xml
./mediatmp/grammar/
./mediatmp/grammar/grammar.xml
```

```
./mediatmp/etc/
./mediatmp/etc/properties.cfg
Delivery file created, can be found
ca_vva.prompt.sv.video1.company.cis0001.r1a.001.mcp0001.t
ar!
```

4 Building an Application Package

4.1 Building Workflow

The building of a Application Package includes the steps described in subsections below. These steps should be performed in the following order:

a Section 4.2 Configure app_properties.cfg on page 11

b Section 4.3 Build the delivery file on page 14

4.2 Configure app_properties.cfg

The app_properties.cfg file defines the characteristics on the Application Package to be built.

1. Change to the <building_dir>/amcpm directory

   ```
   # cd <building_dir>/amcpm
   ```

2. Edit the app_properties.cfg. See for Section 4.2.1 Explanation of the app_properties.cfg file on page 11 details.

3. Save the file and exit the editor.

4.2.1 Explanation of the app_properties.cfg file

The app_properties.cfg contains a number of properties possible to modify according to the description below.

Explanation:

vxmlfiles The name (including path) to the VoiceXML files to be included for this Application Package. The path can be specified using wildcards or by listing the files one by one, e.g

- vxmlfiles=/vobs/ipms/vva/vxml/*.vxml
- vxmlfiles=/vobs/ipms/vva/vxml/myfile.vxml /vobs/ipms/vva/vxml/yourfile.vxml

This property is optional.

ccxmlfiles The name (including path) to the CCXML files to be included for this Application Package. The path can be specified using wildcards or by listing the files one by one, e.g

- ccxmlfiles=/vobs/ipms/vva/ccxml/*.ccxml
- ccxmlfiles=/vobs/ipms/vva/ccxml/service1.ccxml /vobs/ipms/vva/ccxml/service2.cxml

This property is optional.

ecmafiles

The name (including path) to the ECMA files to be included for this Application Package. The path can be specified using wildcards or by listing the files one by one, e.g

- ecmafiles=/vobs/ipms/vva/ecma/*.js
- ecmafiles=/vobs/ipms/vva/ecma/libMyApp.js /vobs/ipms/vva/ecma/libYourApp.js

This property is optional.

eventdefinitionfile

The name (including path) to the Event definition configuration file (note that schema file **must not** be specified here) to be included for this Application Package. This file is an XML file and described in Reference [2] The path can be specified using wildcards or specified explicitly, but it must result in only one file e.g.

- eventfiles=/vobs/ipms/vva/event/eventdefinition.xml

This property is optional.

eventtemplatefiles

The name (including path) to the Event template files configuration files to be included for this Application Package. These files are described in Reference [2] The path can be specified using wildcards or by listing the files one by one, e.g

even

- eventtemplatefiles=/vobs/ipms/vva/event/*.vm
- eventtemplatefiles=/vobs/ipms/vva/eventtemplate/mvioff.vm /vobs/ipms/vva/eventtemplate/slamdown.vm

This property is optional.

numberanalysisfile

The name (including path) to the Number Analysis configuration file (note that schema file **must not** be specified here) to be included for this Application Package. The Number Analysis configuration file is described in Reference [2]. The path can be specified

using wildcards or specified explicitly but it must result in only one file e.g.

- numberanalysisfile=/vobs/ipms/vva/na/numberanalysis.xml

This property is optional.

configfiles

The name (including path) to the configuration files (including schema files) to be included for this Application Package. The path can be specified using wildcards or by listing the files one by one, e.g

- configfiles=vobs/ipms/vva/cfg/vva.x??
- configfiles=/vobs/ipms/vva/cfg/vva1.xml /vobs/ipms/vva/cfg/vva2.xml

This property is optional.

name

The name of the Application Package. This part is used to identify the Application Package as a customer adaptation.

- name=ca_vva

This property is mandatory.

customer

The name of the customer the Application Package is intended for. This part can include both the customer name as well as the CIS number.

- customer=customername.cis0001

This property is mandatory.

productid

The product identifier for the Application Package. This can be used to uniquely identify an Application Package, e.g.

- productid=vva0001/1

This property is mandatory.

rstate

The R-state of the Application Package e.g.

- rstate=p12a_079

This property is mandatory.

service

The services offered by this Application Package. For each serivce offered one entry is added. The following properties shall be specified: The name of the service,

the entry ccxml document that shall be launched when a service request arrives, the protocol to use to access the service and finally the tcp port to use to access the service.

- service=incoming_call incoming_call.ccxml sip 5060

This property is mandatory.

4.2.2 Example on app_properties.cfg file

```
# The properties of the Application Package
vxmlfiles=/vobs/ipms/vva/vxml/*.vxml
ccxmlfiles=/vobs/ipms/vva/ccxml/*.ccxml
ecmafiles=/vobs/ipms/vva/ecma/*.js
configfiles=/vobs/ipms/vva/cfg/vva.x??
eventdefinitionfile=/vobs/ipms/vva/evt/trafficeventsender.xml
eventtemplatefiles=/vobs/ipms/vva/evt/*.vm
numberanalysisfile=/vobs/ipms/vva/na/numberanalyzer.xml
name=ca_vva
customer=company.cis001
productid=vva0001
rstate=r1a.001
service=incomming_call incoming_call.ccxml sip 5060
service=outdial_notification \
outdial_notification.ccxml xmp 9090
```

The deliveryfilename is obtained from this configuration as follows:

<name>.<customer>.<rstate>.<productid>.tar

for example:

```
ca_vva.company.cis001.r1a.001.vva0001.tar
```

4.3 Build the delivery file

a Goto the location where the build script is located:

cd <building_dir>

b Build the delivery file by issuing the following command (if the <apppropertiescfg> argument is left out on the command line, this script tries to use app_properties.cfg file in the current directory if it exists):

./mkapp.sh [<apppropertiescfg>]

c A printout on the terminal shows where the delivery file can be obtained

4.4 Example

Below is shown a simple example of building a Application Package

```
# ./mkapp.sh
Gather Application Package properties ...
About to create ca_vva.company.cis001.r1a_001.vva0001 \
Application Package!
Copy files to temporary directory ...
Make delivery file ...
./apptmp/
./apptmp/vxml/
./apptmp/vxml/doc1.vxml
./apptmp/vxml/doc2.vxml
./apptmp/ccxml/
./apptmp/ccxml/default.ccxml
./apptmp/ccxml/incomming_call.ccxml
./apptmp/ccxml/outdial_notification.ccxml
./apptmp/ccxml/pager_notification.ccxml
./apptmp/ecma/
./apptmp/ecma/libApp.js
./apptmp/cfg/
./apptmp/cfg/vva.xml
./apptmp/cfg/vva.xsd
./apptmp/numberanalysis/
./apptmp/numberanalysis/numberanalyzer.xml
./apptmp/eventdefinition/
./apptmp/eventdefinition/trafficeventsender.xml
./apptmp/eventtemplates/
./apptmp/eventtemplates/cliinformationmessage.vm
./apptmp/eventtemplates/mwioff.vm
./apptmp/eventtemplates/slamdowninformation.vm
./apptmp/etc/
./apptmp/etc/properties.cfg
Cleaning up ...
Delivery file created, can be found \
ca_vva.company.cis001.r1a_001.vva0001.tar!
```

5 Appendix, Writing Grammar Rule files

5.1 Introduction

MAS uses Grammar Rule files that should be defined for each supported language. Grammar rule files define rules for how MAS MUST play natural numbers, date and time.

Number Decomposition support means that a client application can pass a number in various formats (see Decomposition Number Types below) and have that number spoken to the user/listener. The number is not read back digit-by-digit but is read back in a natural format and in the correct gender as appropriate.

For example in US English, "101" won't be voiced as "one zero one", it will be voiced as "one hundred one". In Danish, this will be voiced as "Et hundrede og et" (that is, "one hundred and one").

This section describes how to add number decomposition support and how to write a grammar rule file for a specific language. This is done by creating the appropriate recorded prompts and a language-specific rules file and adding them to the system.

5.1.1 General algorithm

The basic idea is to take the input number (date, time, duration is first converted to an integer number) and apply a sequence of divisions to it to effectively parse out the number and in the process of doing so build up a string representing how you want the number spoken to the listener. This string is a list of sound file names to be played back in sequence. When the rule file is created, the Recording manuscript for this language has to be updated with the possibly new sound file names/prompts found.

The parsing algorithm is generic and makes no assumptions about the structure of the number so no number type or language dependencies are 'hard coded' into MAS. Instead the parsing algorithm is initialized by reading in a set of 'division rules' from a separate XML-file for each language. The algorithm then applies the rules in sequence to the number to decompose it.

Each rule consists of a divisor and value range tests to apply to the results (the quotient and remainder) to determine if the rule succeeded. If the tests succeed, then the result string (the sound file name or names) is returned and is inserted in the current list. The rules are subsetted by divisor and once a rule succeeds within a divisor subset, no more rules are applied to the number from that subset. The number has then been broken down into a quotient, a remainder and a result string. The result string is saved and the quotient and remainder are then subjected to other divisor subsets. The process continues

recursively in this way until the number has been consumed (divided down to 0) - so the result string set (a list of sound file names to be played back to the user) should be complete.

In a couple of special cases, some post-processing is applied to the result list to arrange it in the correct sequence

5.2 Grammar Rule file structure

The rules to use for "converting" numbers to a list of spoken words, are defined in a Grammar Rule file which contains sections for each supported type (date format, 12 and 24 hour time and number). It is also possible to define different rules for different genders (none, female and male) for a type.

The Grammar Rule file uses the following XML structure:

```
<grammar>
    <rule ...>
        <condition ...>
            <action .../>
        </condition>
    </rule>
</grammar>
```

The root node is `<grammar>`.

5.2.1 Rule Element

The Grammar Rule file contains several rule elements.

```
<rule type="Number" gender="None,Male">
    ...
</rule>
```

A rule element contains the rules for a number type. The type is identified by the attribute "type", and the values "Number", "DateDM", "Time12" and "Time24" are valid. The attribute gender may contain one or many genders. Valid entries are "Female", "Male" and "None".

5.2.2 Condition Element

A condition contains a single decomposition rule that should be applied to the input number.

```
<condition divisor="integer"
           atomic="boolean"
           quotientFrom="integer"
           quotientTo="integer"
           remainderFrom="integer"
```

```
            remainderTo="integer"
            terminal="boolean"
            divide="boolean">
    ...
</condition>
```

The conditions properties are defined by its attributes:

- **divisor**

  Number that MUST be used to divide the given number if the corresponding divide field is set to true & atomic field is set to false.

- **atomic**

  MUST be set to true or false.

  true implies that the number shall not be divided further.

  false implies that the number shall be divided further.

- **quotientFrom**

  When the given number is divided by Divisor, the actual quotient MUST be checked to find if it is >= QuotientFrom

- **quotientTo**

  When the given number is divided by Divisor, the actual quotient MUST be checked to find if it is <= QuotientFrom

- **remainderFrom**

  When the given number is divided by Divisor, the actual remainder MUST be checked to find if it is >= RemainderFrom

- **remainderTo**

  When the given number is divided by Divisor, the actual remainder MUST be checked to find if it is <= RemainderFrom

- **terminal**

  MUST be set to true or false

  true implies that if the quotient >= QuotientFrom && the quotient is <= QuotientTo then quotient MUST NOT be processed.

  false implies that if the quotient >= QuotientFrom && the quotient is <= QuotientTo then save the current value of rule index, and the quotient MUST be processed recursively starting from the current rule

- **divide**

MUST be set to true or false.

true means divide the number by the divisor mentioned in this rule.

false means don't divide the number by the divisor mentioned in this rule.

5.2.3 Action Element

The action elements contains the result if the condition succeed (i.e. the quote and remainder is within the specified range). A condition may have several actions. The action element may have the type mediafile, swap, skip or select. A mediafile action element contains a name of a file that should be played, without file extension.

5.2.3.1 Skip Action

A skip action element instructs the post processor to remove the next mediafile element if this is the beginning of the list.

In special cases, it may be required to remove strings from the final output. If the skip element is used, the following mediafile will only be added to the result if there already is a mediafile in the result list. An example of this follows:

```
<condition divisor="1000000" atomic="false"
           quotientFrom="0" quotientTo="0"
           remainderFrom="0" remainderTo="0"
           terminal="true" divide="true">
    <action type="mediafile">0</action>
    <action type="mediafile">seconds</action>
</condition>
<condition divisor="1000000" atomic="false"
           quotientFrom="1" quotientTo="1"
           remainderFrom="0" remainderTo="99"
           terminal="true" divide="true">
    <action type="mediafile">1</action>
    <action type="mediafile">hour</action>
</condition>
<condition divisor="1000000" atomic="false"
           quotientFrom="1" quotientTo="1"
           remainderFrom="100" remainderTo="9900"
           terminal="true" divide="true">
    <action type="mediafile">1</action>
    <action type="mediafile">hour</action>
    <action type="mediafile">and</action>
</condition>
<condition divisor="1000000" atomic="false"
           quotientFrom="1" quotientTo="1"
           remainderFrom="10000" remainderTo="999900"
           terminal="true" divide="true">
    <action type="mediafile">1</action>
```

```
    <action type="mediafile">hour</action>
</condition>
<condition divisor="1000000" atomic="false"
           quotientFrom="2" quotientTo="99"
           remainderFrom="0" remainderTo="99"
           terminal="false" divide="true">
    <action type="mediafile">hours</action>
</condition>
<condition divisor="1000000" atomic="false"
           quotientFrom="2" quotientTo="99"
           remainderFrom="100" remainderTo="9900"
           terminal="false" divide="true">
    <action type="mediafile">hours</action>
    <action type="mediafile">and</action>
</condition>
<condition divisor="1000000" atomic="false"
           quotientFrom="2" quotientTo="99"
           remainderFrom="10000" remainderTo="999900"
           terminal="false" divide="true">
    <action type="mediafile">hours</action>
</condition>
<condition divisor="10000" atomic="false"
           quotientFrom="1" quotientTo="99"
           remainderFrom="0" remainderTo="0"
           terminal="true" divide="false">
    <action type="mediafile">and</action>
</condition>
...
```

Test if the time in question have only minutes and no seconds (pls. note that the table above doesn't show the rules for minutes and seconds). But there's still a problem with this, if we have no hours, then the output for 00:01:00 will be: "and one minute". We add the extra skip action element before "and":

```
<condition divisor="10000" atomic="false"
           quotientFrom="1" quotientTo="99"
           remainderFrom="0" remainderTo="0"
           terminal="true" divide="false">
    <action type="skip"/>
    <action type="mediafile">and</action>
</condition>
```

Now, if "and" is the first mediafile, it will be removed and the result is "one minute".

5.2.3.2 Swap Action

A swap action element instructs the post processor to swap the following mediafile with the mediafile n places to the left where n is the value within the element.

```
<condition divisor="40" atomic="false"
           quotientFrom="1" quotientTo="1"
           remainderFrom="0" remainderTo="0"
           terminal="true" divide="true">
    <action type="mediafile">40</action>
</condition>
<condition divisor="40" atomic="false"
           quotientFrom="1" quotientTo="1"
           remainderFrom="1" remainderTo="9"
           terminal="true" divide="true">
    <action type="mediafile">40</action>
    <action type="mediafile">and</action>
    <action type="swap">-2</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>
```

In this example, if 41 is the input number the output before post processing will be:

```
"40 and (swap -2) 1"
```

But for Danish the output needs to be: "1 and 40" so the swap element tells the post processor to swap the following mediafile (the ’1’) with the mediafile 2 places to the left (since you have the value -2), that is ’40’. So the 1 and the 40 are swapped and you end up with the correct output sequence.

5.2.3.3 Select Action

A select action element desides between two mediafiles and directs the post processor to use one token or the other based on it’s position in the result list. If this is the last in the list, use the mediafile before the select element; otherwise, use the one following the select element.

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

The above example illustrates a case in German, "eins" and "ein". The first is used for the number "one". The second is used when something is following, e.g. "one hundred".

5.2.4 Divisor Rule Subsets

Decomposition rules may be grouped in Divisor Rule Subset.

```
<condition divisor="1000000" atomic="false"
           quotientFrom="0" quotientTo="0"
           remainderFrom="0" remainderTo="0"
           terminal="true" divide="true">
    <action type="mediafile">0</action>
</condition>
<condition divisor="1000000" atomic="false"
           quotientFrom="1" quotientTo="1"
           remainderFrom="0" remainderTo="999999"
           terminal="true" divide="true">
    <action type="mediafile">1</action>
    <action type="mediafile">1e3</action>
</condition>
<condition divisor="1000000" atomic="false"
           quotientFrom="2" quotientTo="999"
           remainderFrom="0" remainderTo="999999"
           terminal="false" divide="true">
    <action type="mediafile">1e3a</action>
</condition>
<condition divisor="1000" atomic="false"
           quotientFrom="1" quotientTo="1"
           remainderFrom="0" remainderTo="999"
           terminal="true" divide="true">
    <action type="mediafile">1</action>
    <action type="mediafile">1e3</action>
</condition>
<condition divisor="1000" atomic="false"
           quotientFrom="2" quotientTo="999"
           remainderFrom="0" remainderTo="999"
           terminal="false" divide="true">
    <action type="mediafile">1e3</action>
</condition>
```

In the exaple abowe, the ’1000000’ begins one divisor subset group and ’1000’ begins the next. Processing of an input number begins with the group with the highest divisor and the division remainder is ultimately passed down to the rest. All conditions tests are applied in a subset until a rule succeeds. When that happens the action is returned and no other tests from the rest of the rules in the subgroup are applied. The remainder (and quotient if it’s greater than 1) is processed by the next subset of rules.

5.2.5 Example

The following rule if succeeded, returns the name of the media file to be used for the spoken word "nineteen"

```
<condition divisor="19" atomic="true"
           quotientFrom="1" quotientTo="1"
           remainderFrom="0" remainderTo="0"
           terminal="true" divide="true">
    <action type="mediafile">19</action>
</condition>
```

A rule 'succeeds' when the division is applied to the number and the quotient is within the quotient range and the remainder is within the remainder range.

In this example, the rule indicates

- to divide the input number by 19
- if the rule succeeds - that is, If the quotient is in the range from 1 to 1 (is exactly equal to 1) AND the remainder is 0 (the number is = 19) then
  - Return the action element "19" which will be appended to the current result list
  - Since 'Divide' is TRUE, consume this input number
  - Since 'Terminal' is TRUE do nothing with the quotient
  - Since 'Atomic' is TRUE, do nothing with the remainder - that is, the number has been totally consumed by this division process.
- If the rule fails, then the number is passed to the next rule

5.3 Number decomposition

This section describes how natural numbers are broken down using division rules starting with the highest divisor supported. See the example for an overview of how this is done.

5.3.1 Limitations/Assumptions

- integers only

- base 10 numbers

- 15 digits maximum

5.3.2 Example

Here's the masculine number rule set. The rules are applied in descending order of divisor.

```
<rule type="Number" gender="Male">
    <condition divisor="1000000000000" atomic="true"
```

```
            quotientFrom="0" quotientTo="0"
            remainderFrom="0" remainderTo="0"
            terminal="true" divide="true">
        <action type="mediafile">0</action>
    </condition>
    <condition divisor="1000000000000" atomic="false"
            quotientFrom="1" quotientTo="1"
            remainderFrom="0" remainderTo="999999999999"
            terminal="true" divide="true">
        <action type="mediafile">1e12</action>
    </condition>
    <condition divisor="1000000000000" atomic="false"
            quotientFrom="2" quotientTo="999"
            remainderFrom="0" remainderTo="999999999999"
            terminal="false" divide="true">
        <action type="mediafile">1e12a</action>
    </condition>
    <condition divisor="1000000000" atomic="false"
            quotientFrom="1" quotientTo="1"
            remainderFrom="0" remainderTo="999999999"
            terminal="true" divide="true">
        <action type="mediafile">1e9</action>
    </condition>
    <condition divisor="1000000000" atomic="false"
            quotientFrom="2" quotientTo="999"
            remainderFrom="0" remainderTo="999999999"
            terminal="false" divide="true">
        <action type="mediafile">1e9a</action>
    </condition>
    <condition divisor="1000000" atomic="false"
            quotientFrom="1" quotientTo="1"
            remainderFrom="0" remainderTo="999999"
            terminal="true" divide="true">
        <action type="mediafile">1e6</action>
    </condition>
    <condition divisor="1000000" atomic="false"
            quotientFrom="2" quotientTo="999"
            remainderFrom="0" remainderTo="999999"
            terminal="false" divide="true">
        <action type="mediafile">1e6a</action>
    </condition>
    <condition divisor="1000" atomic="false"
            quotientFrom="1" quotientTo="999"
            remainderFrom="0" remainderTo="999"
            terminal="false" divide="true">
        <action type="mediafile">1e3</action>
    </condition>
    <condition divisor="100" atomic="false"
            quotientFrom="1" quotientTo="9"
            remainderFrom="0" remainderTo="99"
            terminal="false" divide="true">
```

```
    <action type="mediafile">100</action>
</condition>
<condition divisor="90" atomic="false"
           quotientFrom="1" quotientTo="1"
           remainderFrom="0" remainderTo="9"
           terminal="true" divide="true">
    <action type="mediafile">90</action>
</condition>
<condition divisor="80" atomic="false"
           quotientFrom="1" quotientTo="1"
           remainderFrom="0" remainderTo="9"
           terminal="true" divide="true">
    <action type="mediafile">80</action>
</condition>
<condition divisor="70" atomic="false"
           quotientFrom="1" quotientTo="1"
           remainderFrom="0" remainderTo="9"
           terminal="true" divide="true">
    <action type="mediafile">70</action>
</condition>
...
<condition divisor="19" atomic="true"
           quotientFrom="1" quotientTo="1"
           remainderFrom="0" remainderTo="0"
           terminal="true" divide="true">
    <action type="mediafile">19</action>
</condition>
<condition divisor="18" atomic="true"
           quotientFrom="1" quotientTo="1"
           remainderFrom="0" remainderTo="0"
           terminal="true" divide="true">
    <action type="mediafile">18</action>
</condition>
...
<condition divisor="3" atomic="true"
           quotientFrom="1" quotientTo="1"
           remainderFrom="0" remainderTo="0"
           terminal="true" divide="true">
    <action type="mediafile">3</action>
</condition>
<condition divisor="2" atomic="true"
           quotientFrom="1" quotientTo="1"
           remainderFrom="0" remainderTo="0"
           terminal="true" divide="true">
    <action type="mediafile">2</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>
```

The first rule subset is for extracting the sound file for the largest number supported (if the number is in that range: 1000,000,000,000 - 999,000,000,000,000). The first test condition is actually a special test to see if the number is 0. If so, return "0" and stop processing in this subgroup.

The second test conditions test to see if the number is 1xxx,xxx,xxx,xxx (that is, the quotient is 1 with some remainder). If so, return "1E12" (the singular form of 1000,000,000,000) " and stop processing in this subgroup. Continue processing the remainder with the next subset of rules.

The third test conditions test to see if the number is NNN,xxx,xxx,xxx,xxx (that is, the quotient, NNN, is in the range 2-999 with some remainder).). If so, return "1E12A" (the plural form of 1000,000,000,000)". The processing will decompose the quotient (the false value of the Terminal field instructs the processor to do so) and then continue processing the remainder with the next subset of rules.

The testing proceeds through the rest of the rules in a similar manner until the number is consumed.

5.4 Date (without year) decomposition

To express a date given in DDMM format, first use Mapping of Days and then Mapping of Months. Day and month can be maximum 2 digits each. The input has the form `yyyy-MM-dd`, where the year is ignored.

The number decomposition algorithm for dates performs some pre-processing of the supplied date. Using the given date and month, first form a 4 digits number, then multiply it by 100. That is:

```
n = ( ddMM * 100 )
```

and n has 6 digits: `ddMM00`.

```
<rule type="DateDM" gender="Female,Male,None">
    <condition divisor="10000" atomic="false"
               quotientFrom="1" quotientTo="1"
               remainderFrom="0" remainderTo="999999"
               terminal="true" divide="true">
        <action type="mediafile">dom1</action>
    </condition>
    <condition divisor="10000" atomic="false"
               quotientFrom="2" quotientTo="2"
               remainderFrom="0" remainderTo="999999"
               terminal="true" divide="true">
        <action type="mediafile">dom2</action>
    </condition>
    <condition divisor="10000" atomic="false"
```

```
                quotientFrom="3" quotientTo="3"
                remainderFrom="0" remainderTo="999999"
                terminal="true" divide="true">
            <action type="mediafile">dom3</action>
        </condition>
        ...
        <condition divisor="10000" atomic="false"
                quotientFrom="31" quotientTo="31"
                remainderFrom="0" remainderTo="999999"
                terminal="true" divide="true">
            <action type="mediafile">dom31</action>
        </condition>
        <condition divisor="100" atomic="false"
                quotientFrom="1" quotientTo="1"
                remainderFrom="0" remainderTo="999999"
                terminal="true" divide="true">
            <action type="swap">-1</action>
            <action type="mediafile">jan</action>
        </condition>
        <condition divisor="100" atomic="false"
                quotientFrom="2" quotientTo="2"
                remainderFrom="0" remainderTo="999999"
                terminal="true" divide="true">
            <action type="swap">-1</action>
            <action type="mediafile">feb</action>
        </condition>
        ...
        <condition divisor="100" atomic="false"
                quotientFrom="11" quotientTo="11"
                remainderFrom="0" remainderTo="999999"
                terminal="true" divide="true">
            <action type="swap">-1</action>
            <action type="mediafile">nov</action>
        </condition>
        <condition divisor="100" atomic="false"
                quotientFrom="12" quotientTo="12"
                remainderFrom="0" remainderTo="999999"
                terminal="true" divide="true">
            <action type="swap">-1</action>
            <action type="mediafile">dec</action>
        </condition>
    </rule>
```

In the example above the Divisor is set to 10000 to find the day using the quotient. That is:

`q = ( n / 10000 )` and

`r = ( n % 10000 )`

Use q to find the appropriate word for day DD. Then:

```
n = r

q = ( n / 100 )
```

Use q to find the appropriate word for month MM. In this example, the word for month is then swapped one position to the left, so for n=241100, the output would be "nov dom24" (November 24th).

5.5 Date (with year) decomposition

To express a date given in YYYYDDMM format, first use Mapping of Years then Mapping of Days and then Mapping of Months. Year is maximum 4 digits and day and month can be maximum 2 digits each. The input has the form `yyyy-MM-dd`.

The number decomposition algorithm for dates performs some pre-processing of the supplied date. Using the given year, date and month, first form a 4 digits number, then multiply it by 100. That is:

```
n = ( ddMM * 100 )
```

and n has 6 digits: `ddMM00`.

then multiply this number with 1000000 to get year.

```
n = ( ddMM00 * 1000000 )
```

and n has 10 digits: `yyyyddMM00`

```
<rule type="CompleteDate" gender="Female,Male,None">
    <condition divisor="1000000" atomic="false"
               quotientFrom="2007" quotientTo="2007"
               remainderFrom="0" remainderTo="999999"
               terminal="true" divide="true">
        <action type="mediafile">year2007</action>
    </condition>
    <condition divisor="1000000" atomic="false"
               quotientFrom="2008" quotientTo="2008"
               remainderFrom="0" remainderTo="999999"
               terminal="true" divide="true">
        <action type="mediafile">year2008</action>
    </condition>
    <condition divisor="1000000" atomic="false"
               quotientFrom="2009" quotientTo="2009"
               remainderFrom="0" remainderTo="999999"
               terminal="true" divide="true">
        <action type="mediafile">year2009</action>
    </condition>
    ...
    <condition divisor="10000" atomic="false"
               quotientFrom="1" quotientTo="1"
```

```
                remainderFrom="0" remainderTo="999999"
                terminal="true" divide="true">
            <action type="mediafile">dom1</action>
        </condition>
        <condition divisor="10000" atomic="false"
                quotientFrom="2" quotientTo="2"
                remainderFrom="0" remainderTo="999999"
                terminal="true" divide="true">
            <action type="mediafile">dom2</action>
        </condition>
        <condition divisor="10000" atomic="false"
                quotientFrom="3" quotientTo="3"
                remainderFrom="0" remainderTo="999999"
                terminal="true" divide="true">
            <action type="mediafile">dom3</action>
        </condition>
        ...
        <condition divisor="100" atomic="false"
                quotientFrom="1" quotientTo="1"
                remainderFrom="0" remainderTo="999999"
                terminal="true" divide="true">
            <action type="swap">-2</action>
            <action type="mediafile">jan</action>
        </condition>
        <condition divisor="100" atomic="false"
                quotientFrom="2" quotientTo="2"
                remainderFrom="0" remainderTo="999999"
                terminal="true" divide="true">
            <action type="swap">-2</action>
            <action type="mediafile">feb</action>
        </condition>
        <condition divisor="100" atomic="false"
                quotientFrom="3" quotientTo="3"
                remainderFrom="0" remainderTo="999999"
                terminal="true" divide="true">
            <action type="swap">-2</action>
            <action type="mediafile">mar</action>
        </condition>
        ...
    </rule>
```

In the example above the Divisor is set to 1000000 to find the year using the quotient. That is:

`q = ( n / 1000000 )` and

`r = ( n % 1000000 )`

Then the Divisor is set to 10000 to find the day using the quotient.

`q = ( n / 10000 )` and

```
r = ( n % 10000 )
```

Use q to find the appropriate word for day DD. Then:

```
n = r
```

```
q = ( n / 100 )
```

Use q to find the appropriate word for month MM. In this example, the word for month is then swapped two positions to the left, so for n=2008241100, the output would be "nov dom24 2008" (November 24th, 2008).

5.6 Time decomposition

5.6.1 Time 12 using AM PM

To express the given 24 hours based time in 12 hours based time in words using am & pm, the type should be set to Time12. The time should be specified as a natural number in hours and minutes using `HH:mm:ss`, where the seconds will be ignored.

Hour and Minute can be maximum 2 digits each, and must be expressed in 24 hours format.

The decomposition algorithm for time performs some pre-processing of the supplied time. First it is multiplied with 10000. If hour is greater than 11 also add the value 7000. This mean that the number is an 8 digit number when used in the Time12 section: `hhmm0000` for AM and `hhmm7000` for PM.

```
<rule type="Time12" gender="Female,Male,None">
    <condition divisor="1000000" atomic="false"
               quotientFrom="0" quotientTo="23"
               remainderFrom="0" remainderTo="597000"
               terminal="false" divide="true">
    </condition>
    <condition divisor="1000000" atomic="false"
               quotientFrom="0" quotientTo="23"
               remainderFrom="0" remainderTo="0"
               terminal="true" divide="false">
        <action type="mediafile">oclock</action>
    </condition>
    <condition divisor="10000" atomic="false"
               quotientFrom="1" quotientTo="9"
               remainderFrom="0" remainderTo="7000"
               terminal="false" divide="true">
        <action type="swap">-1</action>
        <action type="mediafile">o</action>
    </condition>
    <condition divisor="10000" atomic="false"
               quotientFrom="10" quotientTo="60"
```

```
            remainderFrom="0" remainderTo="7000"
            terminal="false" divide="true">
</condition>
<condition divisor="100" atomic="false"
            quotientFrom="0" quotientTo="0"
            remainderFrom="0" remainderTo="0"
            terminal="false" divide="true">
    <action type="mediafile">am</action>
</condition>
<condition divisor="70" atomic="false"
            quotientFrom="100" quotientTo="100"
            remainderFrom="0" remainderTo="0"
            terminal="true" divide="true">
    <action type="mediafile">pm</action>
</condition>
<condition divisor="50" atomic="false"
            quotientFrom="1" quotientTo="1"
            remainderFrom="0" remainderTo="9"
            terminal="true" divide="true">
    <action type="mediafile">50</action>
</condition>
<condition divisor="40" atomic="false"
            quotientFrom="1" quotientTo="1"
            remainderFrom="0" remainderTo="9"
            terminal="true" divide="true">
    <action type="mediafile">40</action>
</condition>
<condition divisor="30" atomic="false"
            quotientFrom="1" quotientTo="1"
            remainderFrom="0" remainderTo="9"
            terminal="true" divide="true">
    <action type="mediafile">30</action>
</condition>
<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="19" atomic="true"
            quotientFrom="1" quotientTo="1"
            remainderFrom="0" remainderTo="0"
            terminal="true" divide="true">
    <action type="mediafile">19</action>
</condition>
...
<condition divisor="2" atomic="true"
            quotientFrom="1" quotientTo="1"
            remainderFrom="0" remainderTo="0"
            terminal="true" divide="true">
    <action type="mediafile">2</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>
```

Lets take 13:32 (1332) as the input data. This has to be represented in 12 hour format, which means AM/PM should be played. In order to find AM/PM we need to divide the number by 100 which will give the quotient and from that AM/PM can be found. However if we follow this procedure AM/PM will be added as the first play file in the list. While it should be the last in the list. In order to achieve this AM/PM should be calculated based on the remainder and not on the quotient. Which implies a number should be added to the end of the string that can identifies AM/PM. In the rule file 7000 is added to specify that it is PM and 0000 is added to specify am. Now the input number looks like 01327000. This conversion is performed automatically by the number builder module.

This number is divided first divided by 1000000 which will return hour as quotient. This will be in the range 1 - 12, the quotient should be passed further down to find the exact sound media object. That is why the first line specifies no action. Once the quotient cannot be divided anymore it is added to the list (List= 1) The remainder part 327000 is divided starting with 1000000 and the minutes are obtained and added to the list (List= 1, 30, 2) The last part which is 7000 when divided by 70 will return 100 for pm and is added to the list (List= 1, 30, 2, pm)

5.6.2 Time 24

To express the given 24 hours based time in 24 hours based time is more straight forward, the type Time24 should be used. The time should be specified as a natural number in hours and minutes using `HH:mm:ss`, where the seconds will be ignored.

Hour and Minute can be maximum 2 digits each, and must be expressed in 24 hours format.

The decomposition algorithm for time performs some pre-processing of the supplied time. The number is multiplied with 100. This mean that the number is an 6 digit number when used in the Time24 section: `HHmm00`.

```
<rule type="Time24" gender="Female,Male,None">
    <condition divisor="10000" atomic="false"
               quotientFrom="0" quotientTo="0"
               remainderFrom="0" remainderTo="0"
               terminal="true" divide="true">
        <action type="mediafile">0</action>
        <action type="mediafile">hour</action>
    </condition>
```

```
<condition divisor="10000" atomic="false"
           quotientFrom="0" quotientTo="0"
           remainderFrom="1" remainderTo="5900"
           terminal="true" divide="true">
    <action type="mediafile">o</action>
</condition>
<condition divisor="10000" atomic="false"
           quotientFrom="1" quotientTo="23"
           remainderFrom="0" remainderTo="0"
           terminal="false" divide="true">
    <action type="mediafile">hour</action>
</condition>
<condition divisor="10000" atomic="false"
           quotientFrom="1" quotientTo="23"
           remainderFrom="1" remainderTo="5900"
           terminal="false" divide="true">
</condition>
<condition divisor="100" atomic="false"
           quotientFrom="10" quotientTo="60"
           remainderFrom="0" remainderTo="99"
           terminal="false" divide="true">
</condition>
<condition divisor="100" atomic="false"
           quotientFrom="1" quotientTo="9"
           remainderFrom="0" remainderTo="99"
           terminal="false" divide="true">
    <action type="swap">-1</action>
    <action type="mediafile">o</action>
</condition>
<condition divisor="60" atomic="false"
           quotientFrom="1" quotientTo="1"
           remainderFrom="0" remainderTo="9"
           terminal="true" divide="true">
    <action type="mediafile">60</action>
</condition>
<condition divisor="50" atomic="false"
           quotientFrom="1" quotientTo="1"
           remainderFrom="0" remainderTo="9"
           terminal="true" divide="true">
    <action type="mediafile">50</action>
</condition>
<condition divisor="40" atomic="false"
           quotientFrom="1" quotientTo="1"
           remainderFrom="0" remainderTo="9"
           terminal="true" divide="true">
    <action type="mediafile">40</action>
</condition>
<condition divisor="30" atomic="false"
           quotientFrom="1" quotientTo="1"
           remainderFrom="0" remainderTo="9"
           terminal="true" divide="true">
```

```
        <action type="mediafile">30</action>
    </condition>
    <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="19" atomic="true"
               quotientFrom="1" quotientTo="1"
               remainderFrom="0" remainderTo="0"
               terminal="true" divide="true">
        <action type="mediafile">19</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>
```

5.7 Grammar File Tool

This is a (Solaris) command line tool used to aid in the development of a language grammar rules file. The tool can be found as specified in Section 2.1 Container File on page 3. The tool uses the particular rule file for the language in question and an input data file. The rule file developer can verify by entering a few different inputs in the data file and check that the tool prints the correct result of the conversion on standard out.

Usage:

% ./grammarFileTool.sh <grammar rules file> <input data file>

Example:

% ./grammarFileTool.sh /mediacontentpackage/grammar.xml /mediacontentpackage/input.txt

The input data file contents must follow this syntax:

```
<data>;<type>;<gender>
```

where <data> is the number that should be decomposed, <type> is one of `Number`, `DateDM`, `Time12` or `Time24` and <gender> is one of `Female`, `Male` or `None`. The file should only have one entry on each line. Lines starting with `//` are treated as comments and are ignored by the number builder.

Example input data file:

```
//Example input file for GrammarFileTool
132;Number;Female
3002;Number;None
2006-12-24;DateDM;None
13:32:00;Time12;None
13:32:00;Time24;None
```

The output from the example above whould look something like this (dependent of the grammar file used):

```
132    1 100 30 2
3002   3 1000 2
2006-12-24    dec dom24
13:32:00      1 30 2 pm
13:32:00      13 30 2
```

6 Appendix, Using Text Media Content

Media Content that is text only can be defined in its ows mediacontent file.

This is because the SCE does not automatically create the contents. An example of a text Media Content:

Media Content File

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

Media Object File

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

```
  </mediaobject>
  <mediaobject type="Text" src="videomessage">
     <sourcetext><![CDATA[Video message ]]></sourcetext>
  </mediaobject>
  <mediaobject type="Text" src="from">
     <sourcetext><![CDATA[from ]]></sourcetext>
  </mediaobject>
</mediaobjects>
```

The Media Content in the example above has three qualifiers that are used to decide the appearance of the text message. The content has the attribute returnall set to true, so all instance elements with a condition that is interpreted as true will be added to the resulting message. If the qualifier ’forwarded’ has the value ’true’, the text MediaObject referenced by ’forward’ will be added to the result. The actual text represented by the ’forward’ MediaObject should be defined in a CDATA element in a media object file in this Media Content Package. The CDATA text can for example be "Forwarded " or "Vidarebefodrat ". The value of the ’messageType’ qualifier decides the next part of the message. If the value is ’voice’, the text MediaObject ’voicemessage’ is used. This MediaObject’s text can for example be "Voice message " or "Röstmeddelande ". The next instance is ’from’, which text MediaObject also is defined in the media object file. It can for example be "from " or "från ". The last instance is a MediaObject qualifier containing a name or telephone number of the sender. This is appended to the end of the result. The final result is a list of MediaObjects representing the text, for example "Forwarded Voice Message from John Doe", or "Video message from John Doe", dependent of the values of the input qualifiers. Operation and Maintenance MAS

Reference List

[1] *Installation Guide MAS*, 1/IG MAS 0001 Uen

[2] *Operation and Maintenance MAS*, 1/OM-MAS 0001Uen