

vMRF Configuration Management

Virtual Multimedia Resource Function

User Guide

Copyright

© Ericsson AB 2016, 2017. All rights reserved. No part of this document may be reproduced in any form without the written permission of the copyright owner.

Disclaimer

The contents of this document are subject to revision without notice due to continued progress in methodology, design and manufacturing. Ericsson shall have no liability for any error or damage of any kind resulting from the use of this document.



Contents

1	Emergency and Priority Call Handling	1
1.1	Configuration of Priority Pool	1
2	Manual Scaling	2
2.1	Manual Scaling in OpenStack	2
3	Locking a VM	5
4	Announcement Configuration	6
4.1	Announcement Storage	6
4.2	Basic Announcements	7
4.3	Variable Announcements	9
4.4	Handling of Announcement Files	26
4.5	Convert Existing Audio Files to vMRF Supported Format	27
4.6	Modify the Default Language Code	28
5	vMRF Configuration Export and Import	29
5.1	vMRF Configuration Export	29
5.2	vMRF Configuration Import	30





1 Emergency and Priority Call Handling

A certain amount of processing capacity (priority pool) can be reserved by the vMRF for emergency and priority calls. The `capacityForPriorityCalls` attribute of the *MediaResourceFunction* MO defines a fraction of the total processing capacity and internal resources that is reserved for emergency and priority calls. The attribute is a numeric value, where 1 is 1/1000 fraction of the total processing capacity. The default value is 20.

1.1 Configuration of Priority Pool

This procedure describes how to set the size of the priority pool.

Prerequisites:

- An Ericsson Command-Line Interface (ECLI) session in Exec mode is open.

Steps

1. Navigate to the *MediaResourceFunction* MO and enter configure mode:

```
>ManagedElement=1,MediaResourceFunction=1  
(MediaResourceFunction=1)>configure
```

2. Modify the value of the `capacityForPriorityCalls` attribute:

```
(config-  
MediaResourceFunction=1)>capacityForPriorityCalls=<value>
```

3. Commit the changes:

```
(config-MediaResourceFunction=1)>commit
```

4. The job is completed.



2 Manual Scaling

This section describes how to perform manual scaling in the VNF cluster by increasing or decreasing the number of VMs in a cluster.

2.1 Manual Scaling in OpenStack

This section describes how to perform manual scaling in OpenStack environment.

2.1.1 Scale-out Procedure

Do the following:

Steps

1. Log in to the OpenStack CLI.
2. Use the **heat stack-update** command to increase the size of the VNF, as shown below.

Note: When using the command, all deployment parameters must have the same values as during stack creation. To ensure this, the same `example_environment.yaml` file must be used, and only the `payload_instance_count` parameter needs to be specified. Any deviation from the original values can impact the traffic.

The following example shows how to use the command to increase the size of the VNF while using the `example_environment.yaml` file:

```
heat stack-update <stack_name> -e
example_environment.yaml -f vmrf.yaml -P
payload_instance_count=<new number of VMs>
```

3. Log in to the dashboard and check that the VMs belonging to the VNF are visible in the **Network Topology** view.
4. Check the status of the VNF by running the following command:

```
verify_vmrf_cluster_status.py
```

When a VM is added to the cluster during scale-out, the following VM related MOs are created automatically:



- *MrfInstance* MO
- *ComputeResource* MO
- *SctpEndpoint* MO
- *MrfMediaInterface* MO

2.1.2 Scale-in Procedure

Do the following:

Steps

1. Use the following command to list all the VMs that are part of the scaling domain within the specific VNF:

```
heat output-show <stack_name> resource_names
```

Note the name of the VM you want to delete.

2. Identify the `UUID` of the VM, by using one of the following options:
 - Open the OpenStack Dashboard. The `UUID` and name is shown for each VM.
 - Log in to the OpenStack CLI and use the `nova list` command. The `UUID` and name is shown for each VM in the printout.
3. To minimize traffic impact, the VM you want to delete from the cluster must be locked. Lock the *MrfInstance* MO that represents the VM by using the instructions described in [Locking a VM](#) on page 5.

The *MrfInstance* MO that represents the VM has a reference to the *ComputeResource* MO with the `UUID` identified before.

4. Log in to the OpenStack CLI and use the `heat stack-update` command to decrease the size of the VNF by deleting the selected VM:

```
heat stack-update <stack_name> -e
example_environment.yaml -f vmrf.yaml -P
payload_instance_count=<new number of VMs> -P
payload_scaling_in_list=<Locked VM index>
```

The `<Locked VM index>` is the last number in the name of the selected VM. For example, if the name of the VM is `mrsv-3`, the value of 3 must be used. If the `payload_scaling_in_list` parameter is not specified, the VM with the highest index is deleted automatically.



Note: When using the command, all deployment parameters must have the same values as during stack creation. To ensure this, the same `example_environment.yaml` file must be used, and only the `payload_instance_count` and the `payload_scaling_in_list` parameters need to be specified. Any deviation from the original values can impact the traffic.

5. Log in to the dashboard and check that the VM is not visible in the **Network Topology** view anymore.

When a VM is removed from the cluster during scale-in, the following VM related MOs are deleted automatically:

- *MrfInstance* MO
- *ComputeResource* MO
- *SctpEndpoint* MO
- *MrfMediaInterface* MO



3 Locking a VM

This procedure describes how to lock a VM.

Prerequisites:

- Other upgrade operations are not ongoing, for example, software upgrade.
- An Ericsson Command-Line Interface (ECLI) session in Exec mode is in progress.

Do the following:

Steps

1. Open an SSH connection to the O&M IP address of the vMRF VNF using the following command:

```
ssh <user ID>@<O&M IP address>
```

2. Start a session by issuing the cliss command.
3. Navigate to the *MrfInstance* MO that represents the VM and enter configure mode:

```
>ManagedElement=1,MediaResourceFunction=1,MrfResource=1,MrfInstance=<mrInstanceId>
```

```
(MrfInstance=<mrInstanceId>)>configure
```

4. Modify the value of the `administrativeState` attribute:

```
(config-MrfInstance=<mrInstanceId>)>administrativeState=<LOCKED>
```

The VM is locked immediately and all ongoing traffic from the VM is lost.

5. Commit the changes:

```
(config-MrfInstance=<mrInstanceId>)>commit
```

6. The job is completed.



4 Announcement Configuration

This section describes the procedures for announcement configuration. If the announcement service provided by vMRF is used, this is a mandatory configuration task.

4.1 Announcement Storage

The announcement files can be stored either in an external announcement storage server, or locally, in the vMRF VMs. The storage server solution is recommended for availability reasons. The local storage should be utilized when remote storage is not available.

4.1.1 Announcement Storage Server

The recommended solution for the storage for announcements is a directory placed in a storage server, which is accessible from each VM in the cluster. A vMRF VM connects to the server with sshfs (Secure SHell FileSystem), and mounts a specified directory path. The announcement files needed for playing the announcements are stored under this directory. The announcement storage is enabled by default for vMRF.

The prerequisites for the storage server are the following:

- ssh connection support with public and private keys
- authentication of the ssh connections from the vMRF VMs
- defined user name and ssh public key
- the public key is appended to the authorized keys file using the following command:

```
cat .ssh/<public_key_file_name> >> .ssh/  
<authorized_keys_file_name>
```

- path for the announcement files

Configuration of the announcement storage in the vMRF is performed during instantiation by filling in the relevant HOT template parameters. Following parameters must be defined:

- user name for the announcement storage server
- ssh private key for the username



- IP address of the announcement storage server
- server port of the announcement storage server
- mount directory path
- remote server ssh host key fingerprint

For more information on the HOT template configuration refer to the relevant *deployment instructions*.

4.1.2 Local Storage

If there is no storage server available in the cloud installation, the announcement files can also be stored locally in the vMRF VMs.

In this case, the HOT template parameters for announcement storage must not be defined at deployment. If the parameters are missing, the vMRF VMs will automatically start up configured for local storage of announcement files. Announcement files can be added to the VMs by using the secure copy (`scp`) command to copy the files to the `/cluster/storage/announcements` directory on the active SC. The vMRF will distribute the contents of this directory from the active SC to all VMs.

If local storage is used, the announcement files must be backed up and copied during the upgrade process manually.

4.2 Basic Announcements

Basic announcements are identified by the announcement ID and a language code. The combination of announcement ID and language code form a unique identity for the basic announcement. For multi-language announcements the announcement ID is the same in multiple instances of *BasicAnnouncement* MOs, but the language code is different.

The audio files must be available in the storage server or in the local storage before basic announcements are created. A separate *BasicAnnouncement* MO must be created for each audio file.

The file names and the file paths of the associated audio files are specified at basic announcement creation.

Note: The file names are case sensitive, and special characters, such as space, are not allowed.

4.2.1 Create a Basic Announcement

This procedure describes how to create a *BasicAnnouncement* MO, that represents a single audio announcement file.



Prerequisites

- An Ericsson Command-Line Interface (ECLI) session in Exec mode is open.
- The announcement files are available on the announcement storage server or in the local storage.

Steps

1. In the MOM, navigate to **ManagedElement=1,MediaResourceFunction=1,Announcements=1,BasicAnnouncements=1** and create a *BasicAnnouncement* MO for each audio announcement file.
2. For each created *BasicAnnouncement* MO, define values for the following attributes:
 - **announcementId**

An identity for the basic announcement. The combination of the `announcementId` and `languageCode` attributes form a unique identity for the basic announcement.
 - **basicAnnouncementId**

The value component of the RDN.
 - **fileName**

The name of the announcement file.
 - **filePath**

The relative file path to the directory where the announcement file is stored. The value of this attribute is relative to the `announcement_storage_server_path` attribute in the deployment template. In case of local storage the value of the attribute is relative to `/cluster/storage/announcements`.

Example: `basic/en-GB/`
 - **languageCode**

Language code of the basic announcement. The combination of the `announcementId` and `languageCode` attributes form a unique identity for the basic announcement.

Language code has the format of an RFC 5646 language tag.

Example: `en-GB`



The following attributes are optional:

- duration

The amount of time the announcement is to be played in milliseconds. If the specified duration is greater than the overall length of the announcement, the announcement is repeated until the duration is elapsed. The playing time is also dependent on the value of the iteration attribute. The selected playing time is always the shortest possible alternative.

- iteration

The number of times the announcement is repeated when played to the user.

- userLabel

Label for free use.

4.3 Variable Announcements

Variable announcements are identified by the variable type and the language code. The combination of variable type and language code form a unique identity for the variable announcement. For multi-language announcements the variable type is the same in multiple instances of *VariableAnnouncement* MOs, but the language code is different.

The audio files and logic files have to be available in the storage server or in the local storage before variable announcements are created. A separate *VariableAnnouncement* MO must be created for each variable announcement logic file.

Note: It is not necessary to create *BasicAnnouncement* MOs for the audio files used for variable announcements.

The file names and the file paths of the associated logic files are specified at variable announcement creation.

Note: The file names are case sensitive, and special characters, such as space, are not allowed.



4.3.1 Logic File Type Examples

The following examples show how to create logic files in Lua for variable announcements.

Example Logic File for Variable Announcement Type Digits

```
--The Following is mandatory for a logic file:
--  -define function get_file_list() that returns a table of all an
--  files referenced in the logic file.
--  -define function get_play_list() that takes the logic input str
--  and returns the table of announcement files to play.
local announcement_file_list = {
    [0] = "basic/en-GB/0.wav",
    [1] = "basic/en-GB/1.wav",
    [2] = "basic/en-GB/2.wav",
    [3] = "basic/en-GB/3.wav",
    [4] = "basic/en-GB/4.wav",
    [5] = "basic/en-GB/5.wav",
    [6] = "basic/en-GB/6.wav",
    [7] = "basic/en-GB/7.wav",
    [8] = "basic/en-GB/8.wav",
    [9] = "basic/en-GB/9.wav"
}
function get_file_list()
    file_list = {}
    for i = 0, 9 do
        table.insert(file_list, announcement_file_list[i])
    end
    return file_list
end
function get_play_list(input)
    play_list = {}
    for i = 1, input:len() do
        digit = tonumber(input:sub(i, i))
        if digit then
            table.insert(play_list, announcement_file_list[digit])
        else
            error("Only digits were expected")
        end
    end
    return play_list
end
```

Example Logic File for Announcement Type Date

```
--The Following is mandatory for a logic file:
--  -define function get_file_list() that returns a table of all an
--  files referenced in the logic file.
```



```
--      -define function get_play_list() that takes the logic input s
--      and returns the table of announcement files to play.
local ordered_number_list = {
    [1] = "basic/en-GB/first.wav",
    [2] = "basic/en-GB/second.wav",
    [3] = "basic/en-GB/third.wav",
    [4] = "basic/en-GB/fourth.wav",
    [5] = "basic/en-GB/fifth.wav",
    [6] = "basic/en-GB/sixth.wav",
    [7] = "basic/en-GB/seventh.wav",
    [8] = "basic/en-GB/eighth.wav",
    [9] = "basic/en-GB/ninth.wav",
    [10] = "basic/en-GB/tenth.wav",
    [11] = "basic/en-GB/eleventh.wav",
    [12] = "basic/en-GB/twelfth.wav",
    [13] = "basic/en-GB/thirteenth.wav",
    [14] = "basic/en-GB/fourteenth.wav",
    [15] = "basic/en-GB/fifteenth.wav",
    [16] = "basic/en-GB/sixteenth.wav",
    [17] = "basic/en-GB/seventeenth.wav",
    [18] = "basic/en-GB/eighteenth.wav",
    [19] = "basic/en-GB/nineteenth.wav",
    [20] = "basic/en-GB/twentieth.wav",
    [30] = "basic/en-GB/thirtieth.wav"
}
local number_list = {
    [1] = "basic/en-GB/1.wav",
    [2] = "basic/en-GB/2.wav",
    [3] = "basic/en-GB/3.wav",
    [4] = "basic/en-GB/4.wav",
    [5] = "basic/en-GB/5.wav",
    [6] = "basic/en-GB/6.wav",
    [7] = "basic/en-GB/7.wav",
    [8] = "basic/en-GB/8.wav",
    [9] = "basic/en-GB/9.wav",
    [10] = "basic/en-GB/10.wav",
    [11] = "basic/en-GB/11.wav",
    [12] = "basic/en-GB/12.wav",
    [13] = "basic/en-GB/13.wav",
    [14] = "basic/en-GB/14.wav",
    [15] = "basic/en-GB/15.wav",
    [16] = "basic/en-GB/16.wav",
    [17] = "basic/en-GB/17.wav",
    [18] = "basic/en-GB/18.wav",
    [19] = "basic/en-GB/19.wav",
    [20] = "basic/en-GB/20.wav",
    [30] = "basic/en-GB/30.wav",
    [40] = "basic/en-GB/40.wav",
    [50] = "basic/en-GB/50.wav",
    [60] = "basic/en-GB/60.wav",
    [70] = "basic/en-GB/70.wav",
    [80] = "basic/en-GB/80.wav",
```



```
[90] = "basic/en-GB/90.wav"
}
local month_list = {
  [1] = "basic/en-GB/january.wav",
  [2] = "basic/en-GB/february.wav",
  [3] = "basic/en-GB/march.wav",
  [4] = "basic/en-GB/april.wav",
  [5] = "basic/en-GB/may.wav",
  [6] = "basic/en-GB/june.wav",
  [7] = "basic/en-GB/july.wav",
  [8] = "basic/en-GB/august.wav",
  [9] = "basic/en-GB/september.wav",
  [10] = "basic/en-GB/october.wav",
  [11] = "basic/en-GB/november.wav",
  [12] = "basic/en-GB/december.wav"
}
local word_list = {
  ["of"] = "basic/en-GB/of.wav",
  ["oh"] = "basic/en-GB/oh.wav",
  ["hundred"] = "basic/en-GB/hundred.wav",
  ["thousand"] = "basic/en-GB/thousand.wav"
}
function get_file_list ()
  file_list = {}
  for k,v in pairs(ordered_number_list) do
    table.insert(file_list, v)
  end
  for k,v in pairs(number_list) do
    table.insert(file_list, v)
  end
  for k,v in pairs(month_list) do
    table.insert(file_list, v)
  end
  for k,v in pairs(word_list) do
    table.insert(file_list, v)
  end
  return file_list
end
function get_play_list(input)
  if input:len() ~= 8 then
    error("Input length is not 8")
  end
  century = tonumber(input:sub(1,2))
  first_century_digit = tonumber(input:sub(1,1))
  last_century_digit = tonumber(input:sub(2,2))
  year = tonumber(input:sub(3,4))
  last_year_digit = tonumber(input:sub(4,4))
  month = tonumber(input:sub(5,6))
  day = tonumber(input:sub(7,8))
  audio_file_list = {}
  if day >= 1 and day <= 20 then
    table.insert(audio_file_list, ordered_number_list[day])
  end
end
```




```

elseif day > 20 and day < 30 then
    table.insert(audio_file_list, number_list[20])
    table.insert(audio_file_list, ordered_number_list[day-20])
elseif day == 30 then
    table.insert(audio_file_list, ordered_number_list[day])
elseif day == 31 then
    table.insert(audio_file_list, number_list[30])
    table.insert(audio_file_list, ordered_number_list[1])
else
    error("Day is bigger than 31")
end
table.insert(audio_file_list, word_list["of"])
if month >= 1 and month <= 12 then
    table.insert(audio_file_list, month_list[month])
else
    error("Month is bigger than 12")
end
if last_century_digit == 0 and first_century_digit > 0 then
    table.insert(audio_file_list, number_list[first_century_digit])
    table.insert(audio_file_list, word_list["thousand"])
elseif century >= 1 and century <= 20 then
    table.insert(audio_file_list, number_list[century])
elseif century > 20 and last_century_digit == 0 then
    table.insert(audio_file_list, number_list[century])
elseif century > 20 then
    table.insert(audio_file_list, number_list[first_century_digit])
    table.insert(audio_file_list, number_list[last_century_digit])
end
if last_century_digit ~= 0 then
    if year == 0 then
        table.insert(audio_file_list, word_list["hundred"])
    elseif year < 10 then
        table.insert(audio_file_list, word_list["oh"])
    end
end
if year >= 20 then
    table.insert(audio_file_list, number_list[year-last_year_digit])
end
end
end

```

Example Logic File for Announcement Type Time

```

--The Following is mandatory for a logic file:
--    -define function get_file_list() that returns a table of all
--    files referenced in the logic file.
--    -define function get_play_list() that takes the logic input s
--    and returns the table of announcement files to play.
local ones_list = {
    [1] = "basic/en-GB/1.wav",
    [2] = "basic/en-GB/2.wav",
    [3] = "basic/en-GB/3.wav",

```



```
[4] = "basic/en-GB/4.wav",
[5] = "basic/en-GB/5.wav",
[6] = "basic/en-GB/6.wav",
[7] = "basic/en-GB/7.wav",
[8] = "basic/en-GB/8.wav",
[9] = "basic/en-GB/9.wav",
[10] = "basic/en-GB/10.wav",
[11] = "basic/en-GB/11.wav",
[12] = "basic/en-GB/12.wav",
[13] = "basic/en-GB/13.wav",
[14] = "basic/en-GB/14.wav",
[15] = "basic/en-GB/15.wav",
[16] = "basic/en-GB/16.wav",
[17] = "basic/en-GB/17.wav",
[18] = "basic/en-GB/18.wav",
[19] = "basic/en-GB/19.wav"
}
local tens_list = {
  [2] = "basic/en-GB/20.wav",
  [3] = "basic/en-GB/30.wav",
  [4] = "basic/en-GB/40.wav",
  [5] = "basic/en-GB/50.wav"
}
local oh_ones_list = {
  [0] = "basic/en-GB/HOURS_PRECISELY.wav",
  [1] = "basic/en-GB/OH1.wav",
  [2] = "basic/en-GB/OH2.wav",
  [3] = "basic/en-GB/OH3.wav",
  [4] = "basic/en-GB/OH4.wav",
  [5] = "basic/en-GB/OH5.wav",
  [6] = "basic/en-GB/OH6.wav",
  [7] = "basic/en-GB/OH7.wav",
  [8] = "basic/en-GB/OH8.wav",
  [9] = "basic/en-GB/OH9.wav",
  [10] = "basic/en-GB/HOUR_PRECISELY.wav"
}
function get_file_list()
  file_list = {}
  for i = 1, 19 do
    table.insert(file_list, ones_list[i])
  end
  for i = 2, 5 do
    table.insert(file_list, tens_list[i])
  end
  for i = 0, 10 do
    table.insert(file_list, oh_ones_list[i])
  end
  return file_list
end
function get_play_list(input)
  play_list = {}
  if input:len() ~= 4 then
```



```

        error("Wrong input length")
    end
    if (tonumber(input) == nil) then
        error("Only digits were expected")
    end
    minute_ones = tonumber(input:sub(4, 4))
    minute_tens = tonumber(input:sub(3, 3))
    hour_ones = tonumber(input:sub(2, 2))
    hour_tens = tonumber(input:sub(1, 1))
    if hour_tens + hour_ones == 0 then
        hour_tens = 2
        hour_ones = 4
    end
    if (hour_tens * 10 + hour_ones) > 24 then
        error("Invalid input for hours")
    elseif hour_tens == 2 then
        table.insert(play_list, tens_list[2])
        if hour_ones ~= 0 then
            table.insert(play_list, ones_list[hour_ones])
        end
    else
        hours = hour_tens * 10 + hour_ones
        table.insert(play_list, ones_list[hours])
    end
    if minute_tens > 5 then
        error("Invalid input for minutes")
    elseif minute_tens > 1 then
        table.insert(play_list, tens_list[minute_tens])
        if minute_ones ~= 0 then
            table.insert(play_list, ones_list[minute_ones])
        end
    elseif minute_tens == 1 then
        minutes = minute_tens * 10 + minute_ones
        table.insert(play_list, ones_list[minutes])
    elseif minute_tens + minute_ones == 0 then
        if hour_tens*10 + hour_ones == 1 then
            table.insert(play_list, oh_ones_list[10])
        else
            table.insert(play_list, oh_ones_list[0])
        end
    else
        table.insert(play_list, oh_ones_list[minute_ones])
    end
    return play_list
end

```

Example Logic File for Announcement Type Number

```

--The Following is mandatory for a logic file:
--    -define function get_file_list() that returns a table of all
--    files referenced in the logic file.

```



```
--      -define function get_play_list() that takes the logic input str
--      and returns the table of announcement files to play.
local number_list = {
    [0] = "basic/en-GB/0.wav",
    [1] = "basic/en-GB/1.wav",
    [2] = "basic/en-GB/2.wav",
    [3] = "basic/en-GB/3.wav",
    [4] = "basic/en-GB/4.wav",
    [5] = "basic/en-GB/5.wav",
    [6] = "basic/en-GB/6.wav",
    [7] = "basic/en-GB/7.wav",
    [8] = "basic/en-GB/8.wav",
    [9] = "basic/en-GB/9.wav",
    [10] = "basic/en-GB/10.wav",
    [11] = "basic/en-GB/11.wav",
    [12] = "basic/en-GB/12.wav",
    [13] = "basic/en-GB/13.wav",
    [14] = "basic/en-GB/14.wav",
    [15] = "basic/en-GB/15.wav",
    [16] = "basic/en-GB/16.wav",
    [17] = "basic/en-GB/17.wav",
    [18] = "basic/en-GB/18.wav",
    [19] = "basic/en-GB/19.wav",
    [20] = "basic/en-GB/20.wav",
    [30] = "basic/en-GB/30.wav",
    [40] = "basic/en-GB/40.wav",
    [50] = "basic/en-GB/50.wav",
    [60] = "basic/en-GB/60.wav",
    [70] = "basic/en-GB/70.wav",
    [80] = "basic/en-GB/80.wav",
    [90] = "basic/en-GB/90.wav"
}
local number_big_units_list = {
    [0] = "basic/en-GB/100.wav",
    [1] = "basic/en-GB/1000.wav",
    [2] = "basic/en-GB/million.wav",
    [3] = "basic/en-GB/billion.wav"
}
function get_file_list()
    file_list = {}
    for i = 0, 19 do
        table.insert(file_list, number_list[i])
    end
    for i = 20, 90, 10 do
        table.insert(file_list, number_list[i])
    end
    for i = 0, 3 do
        table.insert(file_list, number_big_units_list[i])
    end
    return file_list
end
function get_play_list(input)
```



```

play_list = {}
if input:len() > 12 or input:len() < 1 then
    error("Input length must be between 1 and 12 digits")
end
if 'number' ~= type(tonumber(input)) then
    error("Input must be a number. The invalid input was: " ..
end
if tonumber(input) < 0 then
    error("Input must be a positive number. The invalid input w
end
-- Remove leading zeros
input = input:match("0*(%d+)")
-- Add zeros in front of the input data in order to have always
-- 12 digits for processing:
padding = (3 - (input:len() % 3) ) % 3
input = string.rep("0",padding) .. input
iterations = (input:len() / 3) - 1
for i = 0, iterations do
    hundreds = tonumber(input:sub(i*3 + 1, i*3 + 1))
    tens = tonumber(input:sub(i*3 + 2, i*3 + 2))
    ones = tonumber(input:sub(i*3 + 3, i*3 + 3))
    -- Special case for zero '0'
    if iterations == 0 and (hundreds == 0 and tens == 0 and ones == 0) then
        table.insert(play_list, number_list[ones])
    end
    -- Add 'hundreds' audio file
    if hundreds ~= 0 then
        table.insert(play_list, number_list[hundreds])
        table.insert(play_list, number_big_units_list[0])
    end
    -- Add 'tens' audio file
    if tens ~= 0 then
        if tens == 1 then
            table.insert(play_list, number_list[10 + ones])
        else
            table.insert(play_list, number_list[10 * tens])
        end
    end
    -- Add 'units' audio file
    if tens ~= 1 and ones ~= 0 then
        table.insert(play_list, number_list[ones])
    end
    -- Add billions, millions or thousands audio file
    if (iterations - i) == 3 then
        table.insert(play_list, number_big_units_list[3])
    elseif ((iterations - i) == 2) and (hundreds ~= 0 or tens ~= 0) then
        table.insert(play_list, number_big_units_list[2])
    elseif ((iterations - i) == 1) and (hundreds ~= 0 or tens ~= 0) then
        table.insert(play_list, number_big_units_list[1])
    end
end
end

```



```
        return play_list
    end
```

Example Logic File for Announcement Type Money

```
--The Following is mandatory for a logic file:
--    -define function get_file_list() that returns a table of all an
--    files referenced in the logic file.
--    -define function get_play_list() that takes the logic input str
--    and returns the table of announcement files to play
local number_list = {
    [0] = "basic/en-GB/0.wav",
    [1] = "basic/en-GB/1.wav",
    [2] = "basic/en-GB/2.wav",
    [3] = "basic/en-GB/3.wav",
    [4] = "basic/en-GB/4.wav",
    [5] = "basic/en-GB/5.wav",
    [6] = "basic/en-GB/6.wav",
    [7] = "basic/en-GB/7.wav",
    [8] = "basic/en-GB/8.wav",
    [9] = "basic/en-GB/9.wav",
    [10] = "basic/en-GB/10.wav",
    [11] = "basic/en-GB/11.wav",
    [12] = "basic/en-GB/12.wav",
    [13] = "basic/en-GB/13.wav",
    [14] = "basic/en-GB/14.wav",
    [15] = "basic/en-GB/15.wav",
    [16] = "basic/en-GB/16.wav",
    [17] = "basic/en-GB/17.wav",
    [18] = "basic/en-GB/18.wav",
    [19] = "basic/en-GB/19.wav",
    [20] = "basic/en-GB/20.wav",
    [30] = "basic/en-GB/30.wav",
    [40] = "basic/en-GB/40.wav",
    [50] = "basic/en-GB/50.wav",
    [60] = "basic/en-GB/60.wav",
    [70] = "basic/en-GB/70.wav",
    [80] = "basic/en-GB/80.wav",
    [90] = "basic/en-GB/90.wav"
}
local number_big_units_list = {
    [0] = "basic/en-GB/100.wav",
    [1] = "basic/en-GB/1000.wav",
    [2] = "basic/en-GB/million.wav",
    [3] = "basic/en-GB/billion.wav"
}
local money_units_list = {
    [0] = "basic/en-GB/POUNDS.wav",
    [1] = "basic/en-GB/POUND.wav",
    [2] = "basic/en-GB/PENCE.wav",
    [3] = "basic/en-GB/PENNY.wav",
```



```

[4] = "basic/en-GB/POUNDS_EXACTLY.wav",
[5] = "basic/en-GB/POUND_EXACTLY.wav",
[6] = "basic/en-GB/NEGATIVE.wav"
}
function get_file_list()
    file_list = {}
    for i = 0, 19 do
        table.insert(file_list, number_list[i])
    end
    for i = 20, 90, 10 do
        table.insert(file_list, number_list[i])
    end
    for i = 0, 3 do
        table.insert(file_list, number_big_units_list[i])
    end
    for i = 0, 6 do
        table.insert(file_list, money_units_list[i])
    end
    return file_list
end
function get_play_list(input)
    play_list = {}
    if input:len() > 14 or input:len() < 1 then
        error("Input length must be between 1 and 14 digits")
    end
    if 'number' ~= type(tonumber(input)) then
        error("Input must be a number. The invalid input was: " .. input)
    end
    -- Check for negative sign, process and remove it before processing
    if input:sub(1,1) == "-" then
        table.insert(play_list, money_units_list[6])
        input = input:sub(2)
    end
    -- Remove leading zeros
    input = input:match("0*(%d+)")
    -- The input is in the smallest monetary unit which is pence. The input
    -- digits are pence, the remaining digits are pounds. The input
    -- with zeros in front of the pounds data and zeros between pounds
    -- in order to have always 3, 6, 9 or 12 or 15 digits for processing
    pence = input:sub(-2,-1)
    pounds = input:sub(1,-3)
    padding = (3 - (pounds:len() % 3)) % 3
    pounds = string.rep("0",padding) .. pounds
    padding = (3 - (pence:len() % 3)) % 3
    pence = string.rep("0",padding) .. pence
    input = pounds .. pence
    iterations = (input:len() / 3) - 1
    for i = 0, iterations do
        hundreds = tonumber(input:sub(i*3 + 1, i*3 + 1))
        tens = tonumber(input:sub(i*3 + 2, i*3 + 2))
        ones = tonumber(input:sub(i*3 + 3, i*3 + 3))
        -- Special case for zero '0'
    end
end

```



```
if iterations == 0 and (hundreds == 0 and tens == 0 and ones
    -- In case of zero remove the 'negative'
    table.remove(play_list)
    table.insert(play_list, number_list[ones])
end
-- Add 'hundreds' audio file
if hundreds ~= 0 then
    table.insert(play_list, number_list[hundreds])
    table.insert(play_list, number_big_units_list[0])
end
-- Add 'tens' audio file
if tens ~= 0 then
    if tens == 1 then
        table.insert(play_list, number_list[10 + ones])
    else
        table.insert(play_list, number_list[10 * tens])
    end
end
-- Add 'units' audio file
if tens ~= 1 and ones ~= 0 then
    table.insert(play_list, number_list[ones])
end
-- Add billions, millions, thousands, pounds or pence audio f
if (iterations - i) == 4 then
    table.insert(play_list, number_big_units_list[3])
elseif ((iterations - i) == 3) and (hundreds ~= 0 or tens ~=
    table.insert(play_list, number_big_units_list[2])
elseif ((iterations - i) == 2) and (hundreds ~= 0 or tens ~=
    table.insert(play_list, number_big_units_list[1])
elseif ((iterations - i) == 1) and (input:len() == 6 and ( hu
    table.insert(play_list, money_units_list[1])
elseif ((iterations - i) == 1) then
    table.insert(play_list, money_units_list[0])
elseif ((iterations - i) == 0) and (hundreds ~= 0 or tens ~=
    if tens ~= 0 or ones > 1 then
        table.insert(play_list, money_units_list[2])
    else
        table.insert(play_list, money_units_list[3])
    end
elseif ((iterations - i) == 0) then
    if (input:len() == 3) then
        -- Case for 0 pence
        table.insert(play_list, money_units_list[2])
    else
        -- Case for 0 pence but having a pound value
        result = table.remove(play_list)
        if string.find(result, "POUNDS") then
            table.insert(play_list, money_units_list[4])
        else
            table.insert(play_list, money_units_list[5])
        end
    end
end
end
```




```

        end
    end
    return play_list
end

```

4.3.2 Create Logic Files

A logic file contains the algorithm logic for a variable announcement in a form that can directly be interpreted. When a play request for a variable announcement is received by the Announcement Service, it uses the logic in the associated algorithm file, together with data sent along with the play request, to determine which output to generate.

As a service, Ericsson can provide logic files for variable announcements, or, optionally, they can be created using the Lua scripting language, based on the examples shown in [Logic File Type Examples](#) on page 10. The Lua version used in vMRF is 5.3.

For more information on the Lua scripting language, refer to <http://www.lua.org>.

Steps

1. Create the logic file for the variable announcement type needed, based on the example files in [Logic File Type Examples](#) on page 10.
2. Validate the logic file using the Logic File Validator (LFV) tool, by issuing the following commands for each logic file:

- `lua logic-file-validator --check-syntax <logic file name>`
- `lua logic-file-validator --execute <logic file name>`
- `lua logic-file-validator --execute <logic file name> <logic input>`

For more information on the LFV tool, see [Logic File Validator Tool](#) on page 22.

3. Copy the logic file to the storage server.

Note: The path of the logic files is needed during the creation of *VariableAnnouncement* MOs.

4.3.2.1 Lua Restrictions in Variable Logic Definition

The following restrictions apply for variable announcement logic files:



- The functions defined in logic files are the following:

get_file_list() This function returns a table with all audio files that are used by the logic file.

get_play_list(input) This function returns a table with all audio files that are played for a specific input.

- The execution environment for the variable logic is restricted to the following:
 - Basic library use is restricted to following functions:
 - pairs
 - ipars
 - tonumber
 - error
 - type
 - String manipulation library is fully available through the name `string`, for example:
 - Table manipulation library is fully available through the name `table`, for example:

```
string.sub(input, 1, 2)
```

```
table.insert(file_list, "file2")
```

4.3.3 Logic File Validator Tool

The Logic File Validator (LFV) tool is used to validate logic files written by the operator without involving Ericsson in the process. It is available in the vMRF delivery package as a single file in the `tools` directory. The tool file name is `logic-file-validator`. In order to use the LFV tool, the Lua interpreter, version 5.2 or higher, needs to be downloaded and installed from <http://www.lua.org>.

To validate a logic file, the operator must verify that both the syntax and the behavior of the logic file is correct. A logic file that is validated by the LFV tool is compatible with the vMRF.

The LFV tool can be used by issuing the `lua logic-file-validator` command in the CLI.

Options without arguments:



-h, --help Prints the help message and exits the LFV tool.

Options with mandatory arguments:

--check-syntax <logic file name>

This option checks whether the logic file syntax is correct and the mandatory functions `get_file_list()` and `get_play_list(input)` exist.

--execute <logic file name>

This option tests logic file execution without input, and prints the list of audio files used by the logic file.

--execute <logic file name> <input>

This option tests logic file execution with the given input. It also prints the list of audio files to be played for the given input.

Note: It is recommended to verify all inputs to test the entire code in the logic file.

In case of errors, an error message with specific information is printed.

4.3.3.1

Check Logic File Syntax

The following procedure describes how to check logic file syntax.

During logic file syntax validation, the LFV tool checks the following:

- There are no Lua syntax errors in the logic file
- The logic file contains mandatory functions required by vMRF

Steps

1. Check logic file syntax with the LFV tool by issuing the following command: `lua logic-file-validator --check-syntax <logic file name>`

Example

The following example shows a successful syntax check:

```
username@hostname:/home/username > lua logic-file-validator --ch
SUCCESS
```

Example

The following example shows checking a logic file with a missing mandatory function:



```
username@hostname:/home/username > lua logic-file-validator --check
lua: logic-file-validator:55: Function 'get_file_list()' not defined
stack traceback:
  [C]: in function 'error'
  logic-file-validator:55: in function 'execute_action'
  logic-file-validator:132: in function 'main'
  logic-file-validator:148: in main chunk
  [C]: in ?
```

4.3.3.2 Verify Logic File Behavior

The following procedure describes how to check logic file behavior.

During logic file behavior check, the LFV tool prints the following information:

- A list of audio files used in the logic file
- A list of audio files to be played for the given input
- Prohibited functions in the logic file, if any

Steps

1. Verify that all audio files used in the logic file are listed in the output of the **lua logic-file-validator --execute <logic file name>** command.

Example

The example below shows the audio files used by the logic file.

```
username@hostname:/home/username > lua logic-file-validator --execute
basic/en-GB/0.wav
basic/en-GB/1.wav
basic/en-GB/2.wav
basic/en-GB/3.wav
basic/en-GB/4.wav
basic/en-GB/5.wav
basic/en-GB/6.wav
basic/en-GB/7.wav
basic/en-GB/8.wav
basic/en-GB/9.wav
```

2. Verify that the output of the **lua logic-file-validator --execute <logic file name> <logic input>** command contains all audio files used for the input.

Example

The example below shows the audio files used in the logic file only for the given input.



```
username@hostname:/home/username > lua logic-file-validator --ex
234567
basic/en-GB/2.wav
basic/en-GB/100.wav
basic/en-GB/30.wav
basic/en-GB/4.wav
basic/en-GB/1000.wav
basic/en-GB/5.wav
basic/en-GB/100.wav
basic/en-GB/60.wav
basic/en-GB/7.wav
```

3. Verify that the logic file does not use prohibited functions.

Example

The following example shows validation error caused by a prohibited function in the logic file.

```
username@hostname:/home/username > lua logic-file-validator --ex
lua: logic-file-validator.lua:73: test/logicFiles/dofile_not_all
stack traceback:
  [C]: in function 'error'
  logic-file-validator.lua:73: in function 'execute_action'
  logic-file-validator.lua:138: in function 'main'
  logic-file-validator.lua:148: in main chunk
  [C]: in ?
```

4.3.4 Create a Variable Announcement

This procedure describes how to create a *VariableAnnouncement* MO, that represents a single variable announcement type.

Prerequisites

- An Ericsson Command-Line Interface (ECLI) session in Exec mode is open.
- The announcement files are available on the announcement storage server or in the local storage.

Steps

1. In the MOM, navigate to **ManagedElement=1,MediaResourceFunction=1,Announcements=1,VariableAnnouncements=1** and create a *VariableAnnouncement* MO for each variable announcement type.
2. For each created *VariableAnnouncement* MO, define values for the following attributes:



- `variableAnnouncementId`

The value component of the RDN.

- `logicFileName`

The name of the variable announcement logic file.

- `logicFilePath`

The relative file path to the directory where the variable announcement logic file is stored. The value of this attribute is relative to the `announcement_storage_server_path` in the deployment template. In case of local storage the value of attribute is relative to `/cluster/storage/announcements`.

Recommended value: `variable/<languageCode>/`

Example: `variable/en-GB/`

- `languageCode`

The language code of the variable announcement.

The combination of the `variableAnnouncementType` and `languageCode` attributes form a unique identity for the variable announcement.

The language code has the format of an RFC 5646 language tag.

Example: `en-GB`

- `variableAnnouncementType`

Type of the variable announcement.

The combination of the `variableAnnouncementType` and `languageCode` attributes form a unique identity for the variable announcement.

The following attribute is optional:

- `userLabel`

Label for free use.

4.4 Handling of Announcement Files

The audio announcement file is created by recording an announcement. The sample rate of the audio announcement has to be 16 kHz and the audio mode Mono. The audio announcement has to be encoded in 16 bit 16 kHz PCM and stored in Waveform Audio File Format (WAV).



When the audio files for basic and variable announcements are stored to the storage server, the recommended directory structure for the stored audio files in the storage server is the following:

```
/announcement_storage/basic/en-GB/
```

When using local storage, a similar directory structure must be created locally to ensure that vMRF can locate announcement audio files:

```
/cluster/storage/announcements/basic/en-GB/
```

When the logic files for variable announcements are stored to the storage server, the recommended directory structure for the stored logic files in the storage server is the following:

```
/announcement_storage/variable/en-GB/
```

When using local storage, a similar directory structure must be created locally to ensure that vMRF can locate variable announcement logic files:

```
/cluster/storage/announcements/variable/en-GB/
```

4.5 Convert Existing Audio Files to vMRF Supported Format

If there are existing audio files which needs to be migrated from native MRF nodes to be used in vMRF, for example in PCMA or PCMU encoded headerless format, they can be converted to vMRF supported WAV 16 bit 16 kHz PCM audio format files using for example Sound eXchange (SoX) audio editing software. SoX is available in many linux distributions and also formacOS and Windows (<https://sourceforge.net/projects/sox/>).

Note: Conversion of PCMA or PCMU encoded (NB quality) audio files with SoX to WAV 16 bit 16 kHz PCM (WB quality) audio format files does not improve the audio quality from NB to WB. To achieve WB quality announcements, the announcements must be recorded in WB quality, and encoded and stored to WAV 16 bit 16 kHz PCM audio format.

An example using SoX from the command line in linux, macOS, or Windows for converting one headerless PCMA audio file to WAV 16 bit 16 kHz PCM audio format file:

```
#sox -c 1 -r 8000 -t al <INPUT_alaw> -r 16000 -b 16 -t wav <OUTPUT_linear_s16_16k>.wav
```

An example using SoX in linux and macOS converting all headerless PCMA audio files in a directory to WAV 16 bit 16 kHz PCM audio format files:

```
#for i in *; do sox -c 1 -r 8000 -t al ./${i} -r 16000 -b 16 -t wav ${i}.wav; done
```



An example for linux and macOS for removing all non-WAV format files in the directory, after conversion of audio files to WAV 16 bit 16 kHz PCM audio format files:

```
#fdel . -type f -not -name '*.wav' -print0 | xargs -0 rm
```

Note: This command will remove all files in the directory that do not have the `wav` file extension.

An example using SoX in Windows for converting all headerless PCMA audio files in a directory to WAV 16 bit 16 kHz PCM audio format files:

```
#for %i in (*) do sox -c 1 -r 8000 -t al .\%i -r 16000 -b 16 -t wav %i.wav
```

An example for removing all files without file extension in the directory, after conversion of audio files to WAV 16 bit 16 kHz PCM audio format files:

```
#del *.*.
```

Note: This command will remove all files in the directory that do not have any file extension.

4.6 Modify the Default Language Code

This procedure describes how to modify the language code of an announcement. The default language code is used as language code for the basic and variable announcements in case the H.248 (or SIP) message does not contain any language code for the requested announcement.

Prerequisites

An Ericsson Command-Line Interface (ECLI) session in Exec mode is open

Steps

1. In the MOM, navigate to `ManagedElement=1,MediaResourceFunction=1,Announcements=1` and modify the `defaultLanguageCode` attribute.

The language code has the format of the RFC 5646 language tag.



5 vMRF Configuration Export and Import

This section describes exporting and importing MO configuration data, certificates, and credentials from and to the vMRF Managed Object Management (MOM) model, as shown in [Figure 1](#).

The exported archive file includes the certificates and credentials installed under the *CertM* MO, with the appropriate metadata to reinstall them on import. The default format of the archive file is `.tar.gz`.

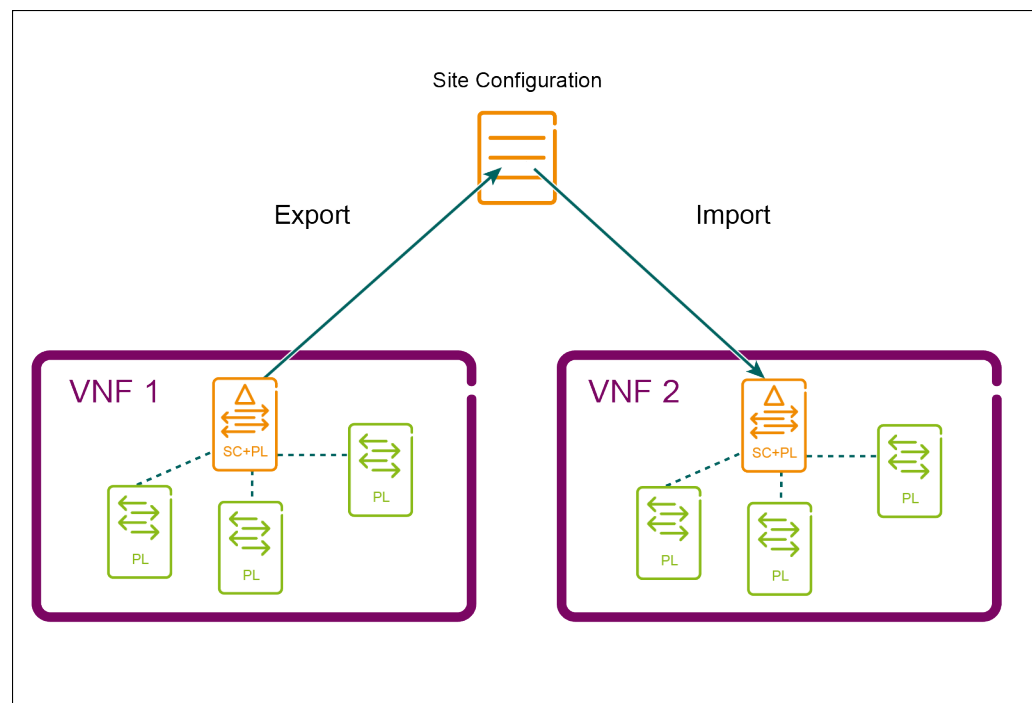


Figure 1 Configuration Export and Import

5.1 vMRF Configuration Export

A configuration export of an existing VNF can be used for backup purposes or as a way of migrating the VNF configuration to a new VNF when performing an upgrade.

Prerequisites:

- A configured and operational vMRF VNF is available and you can log in to it as emergency user, using SSH.

To export the configuration of a vMRF VNF instance, do the following:



Steps

1. Open an SSH connection to the O&M IP address of the VNF instance using the following command:

```
ssh <user ID>@<O&M IP address>
```

2. Export the configuration data in one of the following ways:

Options	Command and result
Export the configuration data into a specified .tar.gz archive file (the default and recommended format)	<p>Use the following command:</p> <pre>/opt/mrf_director/mrf_export_conf.py/ home/<user ID>/ <output_file_without_extension></pre> <p>The path to the output archive file is /home/<user ID>/ <output_file_without_extension>.tar.gz</p>
Export the configuration data with a time stamp as a file name	<p>Use the following command:</p> <pre>/opt/mrf_director/mrf_export_conf.py / home/<user ID>/mrf_conf_{timestamp}</pre> <p>The path to the output archive file is /home/<user ID>/mrf_conf_YYYY-MM-DD_hh-mm-ss.tar.gz</p>

3. If you want to use the exported configuration file in another VNF, copy it out of the file system of the VNF using, for example, the `scp` command:

```
scp <user ID>@<O&M IP address>:/home/<user ID>/  
mrf_conf.tar.gz .
```

Result

The example configuration file `mrf_conf.tar.gz` is copied to the current directory from the `/home/<user ID>/` folder in the file system of the vMRF VNF.

5.2 vMRF Configuration Import

A configuration import can be used for rapid deployment of new VNF instances when initial configuration has been prepared in advance or during an upgrade process to import a previously exported VNF configuration.

Prerequisites:

- An operational vMRF VNF is available and you can log in to it as emergency user, using SSH.



- A file containing vMRF configuration data is available.

To import configuration data from the file, do the following:

Steps

1. If the configuration data file is not available in the file system of the VNF, copy a file to the VNF using, for example, the `scp` command:

```
scp mrf_conf.tar.gz <user ID>@<O&M IP address>:/home/  
<user ID>
```

Result

The configuration file `mrf_conf.tar.gz` is copied from the current directory to the `/home/<user ID>/` folder in the file system of the vMRF VNF.

2. Open an SSH connection to the O&M IP address of the vMRF VNF instance using the following command:

```
ssh <user ID>@<O&M IP address>
```

3. Run the following command:

```
/opt/mrf_director/mrf_import_conf.py /home/<user ID>/  
mrf_conf.tar.gz
```