

Customer Adaptation Guide for Resource Configuration

Ericsson Dynamic Activation 1

USER GUIDE

Copyright

© Ericsson AB 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.

Trademark List

All trademarks mentioned herein are the property of their respective owners. These are shown in the document Trademark Information.



Contents

1	Introduction	1
1.1	Purpose and Scope	1
1.2	Target Groups	1
1.3	Typographic Conventions	1
1.4	Prerequisites	1
2	Resource Configuration Overview	3
2.1	Integrating New Features and Devices	3
2.2	Network Configuration	4
3	Relation Between Feature Instance and Service Instance	7
3.1	Parameter Assignment of ServiceInstanceId	7
4	Feature Models	9
4.1	Model Catalog	9
4.2	Importing New Models	10
5	Importing New Templates	11
5.1	Introduction	11
5.2	Importing and Editing Vendor Templates	13
6	Format of Vendor Template	15
6.1	Template Header	16
6.2	Globals	17
6.3	Before Section	18
6.4	After Section	18
6.5	Feature	19
6.6	Bindings	20
6.6.1	XPath	20
6.6.2	Device Parameter and Credentials	21
6.6.3	Def	22
6.6.4	Match	22
6.6.5	Match All	22
6.7	CLI Commands	23
6.7.1	Request	23
6.7.2	Response	24
6.7.3	Runtime Parameters	25



6.8	NETCONF Commands	25
6.8.1	Request	25
6.8.2	Response	26
6.8.3	Runtime Parameters	26
6.9	SNMP Commands	26
6.9.1	Request	27
6.9.2	Response	27
6.9.3	Runtime Parameters	28
6.10	HTTP Commands	28
6.10.1	Request	28
6.10.2	Response	29
6.11	Repeat	29
6.12	Definitions	30
7	Vendor Template Definitions	31
7.1	Definition Header	31
7.2	Definition Bindings	31
7.2.1	Def	31
8	Vendor Template Transformers	33
8.1	Transformer Header	33
8.2	Transformer Bindings	33
8.2.1	Function	33
9	Template Functions and Primitives	35
9.1	Naming Conventions	35
9.2	Error Handling	35
9.2.1	Parameter Bindings	35
9.3	Parsing Output	36
9.3.1	Matching	36
9.4	Current Configuration	37
9.5	Runtime Parameters	37
9.5.1	Save Runtime Parameter	38
9.5.2	Use Runtime Parameter	38
10	Importing Devices	39
10.1	Import Device JSON Format	39
10.2	Import Device CSV Format (Deprecated)	41
11	Configuring HTTPS Access Point	45
11.1	Additional Parameters	45
	Reference List	47



1 Introduction

This document describes the processes of developing Resource Configuration customizations.

1.1 Purpose and Scope

This document provides an architectural overview for customizations of business logic in Ericsson Dynamic Activation (EDA).

1.2 Target Groups

The target groups for this document are as follows:

- Solution architects
- Solution integrators

1.3 Typographic Conventions

Typographic conventions are described in the *Library Overview*, Reference [1].

1.4 Prerequisites

The readers of this document must meet the following prerequisites:

- Knowledge of Dynamic Activation and Resource Configuration
- Knowledge of Linux
- Knowledge of XML, XML schema, WSDL, and YANG





2 Resource Configuration Overview

Resource Configuration provides a model driven solution for configuration of devices through CLI, NETCONF, HTTP, HTTPS, or SNMP. It makes it possible to integrate network elements during runtime, without the need of programming. The solution is based on the following main components:

- Model Catalog with feature models

The feature model is a logical representation of a network functionality, for example, an Ethernet interface or a BGP configuration in a router. This feature is exposed as a Managed Object (MO) in the northbound CAI3G interface. It provides a uniform way of managing this feature, regardless of what vendor and interface is used to communicate with the actual network southbound. The feature model is defined as an XML schema or as a YANG model.

- Vendor templates

The vendor template is a configuration file in XML format that describes how to setup one or several features in a specific type of device, for example, how to configure the Ethernet interface feature in Juniper routers. The vendor template is the contract defining the command sequence for the required southbound commands.

2.1 Integrating New Features and Devices

It is possible to import additional feature models and templates to extend the supported solutions. This is done through the GUI on the runtime system. It is also possible to edit existing templates from this GUI. When new models and templates have been imported or updated, they take effect on the runtime system.

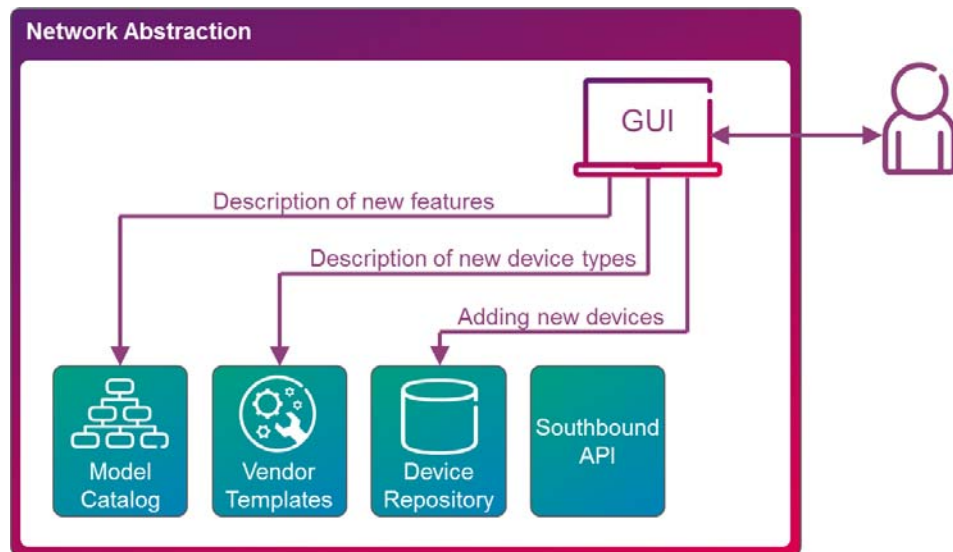


Figure 1 Integrating New Features and Devices

Once the models and templates are in place it is time to register the devices that Resource Configuration manages. This is done through the GUI or a device import file if there is a need to import large bulk of devices at the same time.

The device repository keeps information about the device. For example, it stores information about the device type, which is used to resolve which templates that are applicable for the device.

2.2 Network Configuration

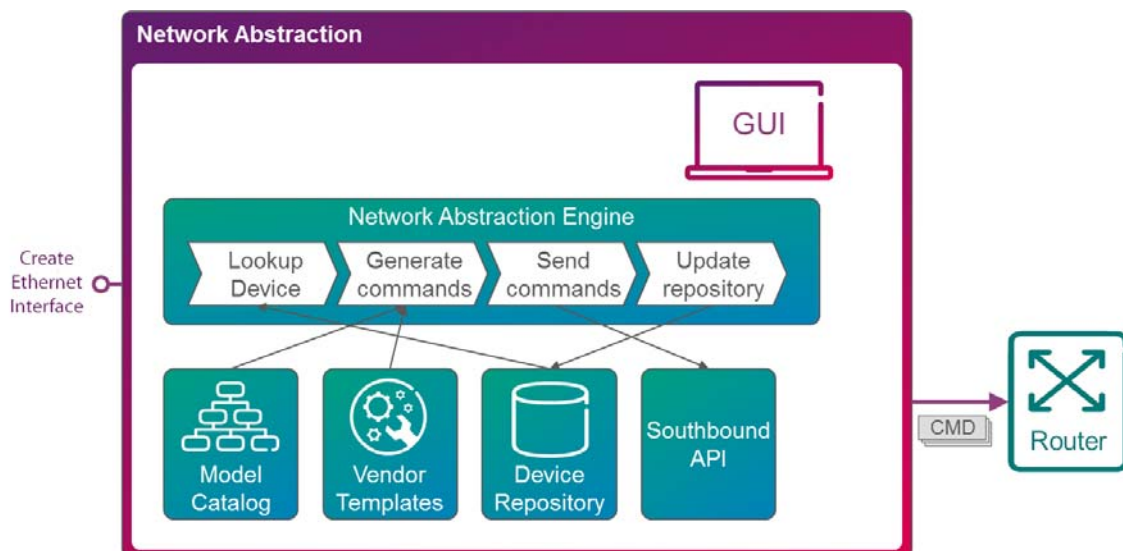


Figure 2 Network Configuration



When a request is sent to Resource Configuration to set up a network feature, for example an Ethernet interface in a Juniper router, the following steps take place:

1. **Lookup device:** based on the information in the incoming request, the Network Abstraction Engine looks up information about the device in the Device Repository. This includes for example IP access points, device parameters and a list of templates.
2. **Generate commands:** the Network Abstraction Engine reads the relevant feature model definition and validates the request.
 - Based on the information in the Device Repository, the correct vendor template is retrieved and the corresponding commands are generated.
 - Based on the template protocol type, the matching access point is used to establish a connection towards the device.

Note: One device can be configured with multiple access points using the same template protocol type. In this scenario, the secure protocols always have a higher priority to be chosen.

For example:

- A CLI template can be used towards either an SSH CLI or a TELNET access point, where SSH CLI has a higher priority.
 - An HTTP template can be used towards either an HTTPS or an HTTP access point, where HTTPS has a higher priority.
3. **Send commands:** Network Abstraction Engine sends the commands to devices via southbound interfaces.
 4. **Update repository:** after the commands are sent, Device Repository is updated with the current configuration of the device.





3 Relation Between Feature Instance and Service Instance

Relating a feature instance to a service instance is done by using the `serviceInstanceId` which by default is exposed as an optional MOId for all feature models in Resource Configuration. A `serviceInstanceId` MOId is automatically added to the feature model wsdl file during interface model creation. For more information about interface model creation, refer to *User Guide for Designer Studio*, Reference [3].

A northbound Create feature request containing a `serviceInstanceId` means that the feature gets related to a service instance. A feature instance related to a service instance requires the `serviceInstanceId` for Set, Get and Delete operations. The `serviceInstanceId` is required to follow the following format:

`{MO Namespace}MO Name##keys`

Where keys are one or more service MOIds. Multiple MOIds are separated by “...”

```
<serviceInstanceId>
  {http://schemas.ericsson.com/ma/CA/L3VPN/}L3VPN##serviceId=1234
</serviceInstanceId>
```

Example 1

```
<serviceInstanceId>
  {http://schemas.ericsson.com/ma/CA/L3VPN/}L3VPN##serviceId1=1234;
</serviceInstanceId>
```

Example 2

3.1 Parameter Assignment of ServiceInstanceId

When developing Service Model in Designer Studio, assign `serviceInstanceId` for each task as follows:

1. Select the **Transformation** option when configuring the parameter assignment of `serviceInstanceId`.



Designer Studio / L3VPN

Version EDA1 Modified, May

Create Get Set

Create task

Task Dependencies Graph

Graph List

Northbound

PE1CreateBGP

6/6 Mandatory

PE1CreateBGP

Type to Search

CreateL3VPN

CreateL3VPN > CE1

CreateL3VPN > CE1 > LoopBack

CreateL3VPN > CE1 > LoopBack > networkid

CreateL3VPN > CE1 > BGP

CreateL3VPN > CE1 > BGP > vrf

CreateL3VPN > PE1

CreateL3VPN > PE1 > LoopBack

CreateL3VPN > PE1 > LoopBack > networkid

CreateL3VPN > PE1 > MPLS

CreateBGP

CreateBGP > neighbors > neighbor

CreateBGP > redistributes > redistribute

CreateBGP > vrfs > vrf

CreateBGP > vrfs > vrf > neighbors > neighbor

CreateBGP > vrfs > vrf > redistributes > redistribute

serviceInstanceid

Transformation

Source value

Source value with default

Fixed value

Transformation

Conditions are not supported for transformations

2. Use CONCAT function to define a transformation expression. For example:

`CONCAT("{http://schemas.ericsson.com/ma/CA/L3VPN/}L3VPN##serviceId", serviceId)`

Designer Studio / L3VPN / Transformation

Transformation

serviceInstanceid

Model: L3VPN | Version: EDA1 | Task: PE1CreateBGP

< Save

Expression

CONCAT("{http://schemas.ericsson.com/ma/CA/L3VPN/}L3VPN##serviceId", serviceId) Test expression ?

Sources

+ Add new source

Source Na...	Task Na...	Path	Test Value
serviceid	Northbo...	CreateL3VPN > servi...	<input type="text"/>

Delete source

For more information about Parameter Assignment, refer to *User Guide for Designer Studio*, Reference [3].



4 Feature Models

4.1 Model Catalog

The Model Catalog is one of the core components of Resource Configuration. It tracks the different feature models managed by the solution. Resource Configuration provides several feature models out-of-the-box that can be modified to suite the needs of customers,, and it is also possible to add more models to the solution. The available models are:

- Access
- Border Gateway Protocol (BGP)
- Bridge Domain
- DHCP Relay Agent
- DHCP Server
- Ethernet Interface
- EVCBridge
- Firewall
- Gateway Load Balancing Protocol GLBP
- Internal Network to Network Interface (INNI)
- IP Protocol Independent Multicast (IPPM)
- Internet
- Loopback Interface
- Multiprotocol Label Switching (MPLS)
- NVE Interface
- Open Shortest Path First (OSPF)
- Policy
- Serial Interface
- System Contact
- TVConnection
- User Network Interface (UNI)



- Virtual Private LAN Service (VPLS)
- Virtual Routing and Forwarding (VRF)
- Virtual Routing VRRP
- Video on Demand (VoD)
- Voice Over IP (VoIP)
- xConnect
- Tunnel

On top of the feature models exposed in the Model Catalog, it is possible to build customer specific service logic in Designer Studio. For more information about how to do this, refer to *User Guide for Designer Studio*, Reference [3].

4.2 Importing New Models

It is possible to import new feature models in the Resource Configuration GUI. When the model has been imported, an interface model can be created. For more information, refer to *User Guide for Resource Configuration*, Reference [2].

After the interface has been created, a corresponding WSDL file will be available for the model. The WSDL file describes how to send northbound requests to Resource Configuration. To fully support all the features in CAI3G, the CAI3G specification put restrictions on the data model. The interface creator cannot add additional information to an imported model. Therefore CAI3G specific functionality, for example SubMOs, must be declared in the imported model if such functionality is required in the interface. For more information, refer to *Provisioning over CAI3G for Resource Configuration*, Reference [4].



5 Importing New Templates

5.1 Introduction

Resource Configuration provides a model driven solution that makes it possible to map from the generic features to vendor specific implementations. The solution will automatically generate the CLI, NETCONF, HTTP, or SNMP commands that are to be sent southbound.

The solution is based on the feature models in the Model Catalog together with vendor templates that defines the structure of the southbound commands. The vendor templates define the request/response behavior for a specific type of device and is provided in XML format. The XML Templates are protocol type specific. This means that the template declares which southbound protocol to be used towards the device, see further information in Section 2.2 on page 4. It will replace the place holder for attribute value with the real attribute values provided from the northbound interface.

Figure 3 shows an example of a vendor template.



```
<?xml version="1.0" encoding="utf-8"?>
<cli_template xmlns="http://schemas.ericsson.com/nca/template/">
  <name>IOS_Template</name>
  <description>Communication template for IOS operating system for cisco devices</description>
  <prompt>">"</prompt>

  <definitions>
    <definition>SCM</definition>
  </definitions>

  <feature name="EthernetInterface">
    <!-- EthernetInterface Model version 2.0 -->
    <operation type="create">
      <bindings>
        <val name="interfaceType">
          <xpath>interfaceType</xpath>
        </val>
        <val name="interfaceNumberWithSub">
          <xpath>interfaceNumber</xpath>
        </val>
        <val name="(interfaceNumber,subInterface)">
          <def>
            {
              interfaceNumberWithSub.split("\\.") match {
                case Array(x) => (x, "")
                case Array(x, y) => (x, y)
                case _ => ("", "")
              }
            }
          </def>
        </val>
      </bindings>
      <command>
        <request>
          (subInterface) match {
            case "" => s"interface $interfaceType$interfaceNumber"
            case _ => s"interface $interfaceType$interfaceNumberWithSub"
          }
        </request>
      </command>
      <command>
        <request>
          "" match {
            case "s" => s"$serviceInstanceId $vlanId"
            case _ => ""
          }
        </request>
      </command>
    </operation>
  </feature>
</cli_template>
```

a

b

c

d

Figure 3 Vendor Template Example

- a The protocol type definition of the vendor template.



- b The relation to the feature in the Model Catalog.
- c The vendor template explains the contract how to communicate over this interface as a sequence of requests and responses.
- d Data values will be provided from the request in the northbound interface and mapped to the generated CLI command according to the mapping from service model to the feature model in the Model Catalog.

Note: The vendor template is not only used when processing requests from the northbound systems. It is also used by a lot of the internal functions in Resource Configuration. A consequence of this is that the template always must provide a mapping for all the different type of operations: create, delete, set, get, and getall.

5.2 Importing and Editing Vendor Templates

It is possible to import and modify vendor templates on the runtime system via the GUI. For more information, refer to *User Guide for Resource Configuration*, Reference [2] .





6 Format of Vendor Template

This section gives an overview of a vendor template.

Examples of templates are included in the delivery of Resource Configuration. The included example files are the following:

- `template.xsd` - Describes the detailed definition of the template format. See Table 1 for information about the layout of the template.
- `Template20.xsd` - Describes the detailed definition of the deprecated legacy template format. See Table 1 for information about the layout of the template.
- `Definition20.xsd` - Describes the detailed definition of the template definition format. See Section 7 on page 31 for information about the layout of the template.
- `Resource_Configuration_Template20_Example` - Describes the template format and the supported primitives and functions for devices using protocol `SSH_CLI`. This file showcases the different possibilities in the templates. For more information about CLI commands in the template see Section 6.7 on page 22.
- `Resource_Configuration_Template20_Example_SNMP.xml` - Describes the template format and the supported primitives and functions for devices using protocol `SNMP`. This file showcases the different possibilities in the templates. For more information about SNMP commands in the template see Section 6.9 on page 26.
- `Resource_Configuration_Template20_Example_HTTP.xml` - Describes the template format and the supported primitives and functions for devices using protocol `HTTP`. This file showcases the different possibilities in the templates. For more information about HTTP commands in the template see Section 6.10 on page 28.

To get the example templates, follow below instruction:

1. Save the zip file, [Resource_Configuration_examples.zip](#) to a temporary location.
2. Unpack the zip file.

When unzipping, there are two folders: `Templates` and `Device Import`.

3. Get the templates from the `Templates` folder and add them in a suitable location.



Table 1 Template Layout

Attribute	Cardinality	Description
name	Mandatory	This is the unique name for the vendor template.
description	Optional	Descriptive text regarding the vendor template.
prompt	1 to many	Specifies what Resource Configuration expects after establishing a connection with the device before sending the first command. For example, <code><prompt>#</prompt></code>
definitions	Optional	A list of referenced Definition names.
globals	Optional	Contains global definitions. For example a response structure which can be reused throughout the template to simplify and avoid repetition.
before	Optional	Defines a set of commands to be executed before each operation.
after	Optional	Defines a set of commands to be executed after each operation.
feature	1 to many	A defined command sequence of southbound commands to set up a specific feature in a device.

The vendor template is based on information in the following types of sections:

- Template header
- Globals
- Before section
- After section
- Feature mappings

The information in these sections is described in more detail in the following sections.

6.1 Template Header

The template header contains descriptive and identifying information about the template.

```
<cli-template xmlns="http://schemas.ericsson.com/nca/template/">  
  <name>TEMPLATE_UNIQUE_NAME</name>  
  <description>TEMPLATE_DESCRIPTION</description>  
  <prompt>PROMPT_TO_READ_AFTER_LOGIN</prompt>
```



The root tag of the template specifies the protocol type to be used:

- `<cli-template>` supports CLI. This protocol type can be used towards both SSH CLI and TELNET access points, where SSH CLI has a higher priority.
- `<netconf-template>` supports NETCONF.
- `<http-template>` supports HTTP. This protocol type can be used towards both HTTPS and HTTP access points, where HTTPS has a higher priority.
- `<snmp-template>` supports SNMP.

6.2 Globals

The vendor template allows a global setting for handling general faults and for simplifying the template. The global setting can hold a response structure which can be used as a default response. Unless otherwise specified in the request, the template uses this default response to resolve whether a command was successful. In this way, there is no need to define this error condition after each command, which facilitates creating and maintaining the vendor template.

It is also possible to use parts of the default response. For example, a request reuses the assertion defined in the `globals` section, but in the request a special case prompt is specified. This means that the assertion information from the `globals` section and the prompt information from the request is used to validate the response.

This global definition should be inserted just after the `<prompt>` tag in the vendor template:

```
<globals>
    <prompt>#</prompt>
    <validate>...</validate>
</globals>
```

The `globals` section in the vendor template consists of the following attributes:

*Table 2 Attributes in Globals Section*

	Cardinality	Description
bindings	Optional	Bindings are used to define global variables that are accessible throughout the entire Template. For more information, see Section 6.6 on page 20.
prompt	Optional	The interpreter will wait until the prompt has been read before the next command is executed.
validate	Optional	Assertions to evaluate the response to determine the success of the request.

6.3 Before Section

The operations, defined in the `before` section, are executed before all regular feature operations, except when the attribute `skipBefore` is set.

The operation executed inside the `before` section corresponds to the operation type of the request. For example, if the request specifies `CreateEthernetInterface` the `create` operation of the `before` section is executed.

For more information regarding the XML structure, see Section 6.5 on page 18.

6.4 After Section

The operations defined in the `after` section are executed after all regular feature operations, except when the attribute `skipAfter` is set.

The operation executed inside the `after` section corresponds to the operation type of the request. For example, if the request specifies `CreateEthernetInterface` the `create` operation of the `after` section will be executed.

For more information regarding the XML structure, see Section 6.5 on page 18.



6.5 Feature

Each template can map one or several features in the Model Catalog. Each feature is mapped by name in its own feature section of the vendor template. The main purpose is to define the sequence of southbound commands necessary to configure a feature in a device. The command sequences are divided into operational (create, delete, set, get and get all) subsections to clarify which sequence of commands are sent.

```
<feature name="FEATURE">
  <operation type="create|delete|set|get|getall">
    <bindings>
      ...
    </bindings>
    <command>
      ...
    </command>
    ...
  </operation>
</feature>
```

The `Feature` section in the vendor template consists of the following attributes:

Table 3 Feature

Attribute	Cardinality	Description
name	Mandatory	This name is directly mapped toward a model in the Model Catalog.
operation	1 to 5	Defines which operations are supported for this feature. Operation types are; create, delete, set, get, and getall. Only one of each operation is allowed to be defined for a feature.
skipBefore	Optional	Enables or disables the execution of <code>Before</code> section for a specific operation. Default set to false.
skipAfter	Optional	Enables or disables the execution of <code>After</code> section for a specific operation. Default set to false.



Attribute		Cardinality	Description
	bindings	Optional	Bindings are used to define local variables. Variable bindings at this level are local for one operation. For more information regarding bindings, see Section 6.6 on page 20.
	command	0 to many	The sequence of commands to be executed to configure the feature. For more information about commands, see Section 6.7 on page 22. At least one of <code>command</code> or <code>repeat</code> is required.
	repeat	0 to many	Defines a sequence of commands to execute for each item in a provided list. For more information regarding repeats, see Section 6.11 on page 29. At least one of <code>command</code> or <code>repeat</code> is required.

6.6 Bindings

Bindings are used to define variables to be used throughout the template. Bindings are provided to simplify the template and also grant access to device-specific parameters, content from the inbound requests, and static values.

6.6.1 XPath

XPaths map parameters in the XML request to local variables in the template for easy access to their values.

The example below defines the local variable `localVariable` to the parameter `Parameter` inside the `Element` structure from the inbound request. This parameter must be present in the request.



```
<bindings>
  <val name="localVariable">
    <xpath>Element/Parameter</xpath>
  </val>
</bindings>
```

Example 3 Mandatory XPath Variable

If the parameter is optional in the model, it is possible to provide a default value for the variable to avoid depending on input from the inbound request. If the parameter is not present in the request, the default value is used.

```
<bindings>
  <val name="localVariable" defaultValue="0">
    <xpath>Element/Parameter</xpath>
  </val>
</bindings>
```

Example 4 Optional XPath Attribute with Default Value

If the parameter is optional in the model, not included in the request, and the attribute "optional" is set to `true`, the variable value is set to `MISSING`.

If the parameter is included in the request, the variable value is set to the parameter value in the request.

```
<bindings>
  <val name="localVariable" optional="true">
    <xpath>Element/Parameter</xpath>
  </val>
</bindings>
```

Example 5 Optional XPath Attribute "optional"

6.6.2 Device Parameter and Credentials

Device parameters can be defined when there is information that is common for all devices of the same type. The template can use these parameters as input. Put the name of the device parameter in the bindings and Resource Configuration extracts the value of that parameter and set it in the variable.

Device parameters can be used when the template should use a parameter value defined in the Device Repository instead of the parameter values it gets from the inbound CAI3G request. The template can use parameters defined as `Parameter` and `Credential` in Device Repository as input. Put the name of the device parameter in the bindings and Resource Configuration extracts the value of that parameter and set it in the variable. For more information about how to define `Parameter` and `Credential` values in the Device Repository, refer to *User Guide for Resource Configuration*, Reference [2].



Example

```
<bindings>
  <val name="localVariable">
    <deviceParameter>DeviceParameterName</
    deviceParameter>
  </val>
  <val name="localVariable2">
    <deviceCredential>DeviceCredentialName</
    deviceCredential>
  </val>
</bindings>
```

6.6.3 Def

Def can be used to define static values.

```
<bindings>
  <val name="localVariable">
    <def>10</def>
  </val>
</bindings>
```

6.6.4 Match

Match map parameters in the request as strings in local variables in the template with regular expression string matching.

```
<val name="localVariable">
  <match>local ([\d.]* [\d.]*)</match>
</val>
```

6.6.5 Match All

MatchAll map parameters in the request as lists of strings in local variables in the template with regular expression string matching.

```
<val name="localVariables">
  <matchAll>local list ([\d.]*) ([\d.]*)</matchAll>
</val>
```



6.7 CLI Commands

The following command structure defines what is sent to the device when using CLI:

```
<command>
  <request>"COMMAND"</request>
  <prompt>"PROMPT"</prompt>
  <validate>...</validate>
</command>
```

The command descriptions contains the following information:

Table 4 Command Definitions

Attribute	Cardinality	Description
request	Mandatory	The request to be sent.
prompt	Optional	The interpreter will wait until the prompt has been read before the next command is executed.
validate	0 or more	Assertions to evaluate the response to determine the success of the request.
response	Optional	Defines how the response is converted to a result document.
bindings	Optional	Bindings are used to define local variables.
result	Mandatory	This is where the data-structure to be returned is defined.
runtimeParameters	Optional	Used to store data from one command for use in a subsequent command in the same operation.

6.7.1 Request

The request can be sent as a plain string:

```
<request>"string"</request>
```

In a request which includes a parameter, all parameters are preceded by a \$ sign:

```
<request>s"string $localVariable"</request>
```



Below shows a CLI example of a request evaluating the value of a parameter and declaring different outcomes based on the value:

```
<request>
  (localVariable) match {
    case "xyz" => s"xyz command $localVariable"
    case "abc" => s"$localVariable abc command"
    case _ => s"sending default command"
  }
</request>
```

6.7.2 Response

This is where the response is converted to a result document.

The following is an example when using CLI:

```
<response>
  <bindings>
    <val name="name type number">
      <match>(\w*?) (\S*) (\d*)</match>
    </val>
  </bindings>
  <result xmlns:any="http://any.namespace.com/any/namespace/">
    <any:name>{name}</any:name>
    <any:type>{type}</any:type>
    <any:number>{number}</any:number>
  </result>
  <runtimeParameter></runtimeParameter>
</response>
```

6.7.2.1 Bindings

Variable bindings at this level are local for this filter and are by default used with response string data instead of request XML data. For more information regarding bindings, see Section 6.6 on page 20. For more information regarding matching and parsing output, see Section 9.3 on page 36.

6.7.2.2 Result

It is important that the response data is compliant with the interface schema the feature-operation is connected to.



6.7.3 Runtime Parameters

Runtime parameters are used to store data from one command for use in a subsequent command in the same operation. For more information regarding Runtime parameters, see Section 9.5 on page 37.

6.8 NETCONF Commands

The following command structure defines what is sent to the device when using NETCONF:

```
<command>
  <request>
    (Netconf xml)
  </request>
  <prompt>"#"</prompt>
  <validate should="match">"OK"</validate>
</command>
```

Note: In a typical NETCONF communication, there is no prompt available, but the vendor template still requires a `<prompt>` tag to be defined. This tag will not be used when generating the southbound commands, but it has to be present to get a valid template. The `<prompt>` tag can therefore contain any dummy information.

For command descriptions, see Table 4.

6.8.1 Request

Below shows a NETCONF example with parameters:

```
<request>
  <configuration>
    <xyz>
      <name>{localVariable}</name>
      <description>{description}</description>
    </xyz>
  </configuration>
</request>
```

Below shows a NETCONF get example:

```
<request>
  <filter type="subtree">
    <configuration>
      <xyz>
        <name>{localVariable}</name>
```



```
        </xyz>
      </configuration>
    </filter>
  </request>
```

6.8.2 Response

This is where the response is converted to a result document.

The following is an example when using NETCONF:

```
<response>
  <bindings>
    <val name="name">
      <xpath>data/configuration/xyz/name</xpath>
    </val>
    <val name="description">
      <xpath>data/configuration/xyz/description</xpath>
    </val>
  </bindings>
  <result xmlns:any="http://any.namespace.com/any/namespace/">
    <any:name>{name}</any:name>
    <any:type>{description}</any:type>
  </result>
</response>
```

6.8.3 Runtime Parameters

Runtime parameters are used to store data from one command for use in a subsequent command in the same operation. For more information regarding Runtime parameters, see Section 9.5 on page 37.

6.9 SNMP Commands

The following command structure defines what is sent to the device when using SNMP. Request is defined in XML format, however Resource Configuration converts it into SNMP protocol message:

```
<command>
  <request>
    (<snmpGet ... /> or <snmpSet ...>...</snmpSet>)
  </request>
</command>
```

For command descriptions, see Table 4.



6.9.1 Request

In the examples in this section, supported values for `version` attribute are "1" and "2".

Below shows an SNMP get example:

```
<request>
  <snmpGet oid=".1.3.6.1.2.1.1.4.0" community="public"
    version="2" />
</request>
```

Below shows an SNMP set example:

```
<request>
  <snmpSet oid=".1.3.6.1.2.1.1.4.0" community="public"
    version="2">test</snmpSet>
</request>
```

Below shows SNMP set example with parameters:

```
<request>
  <snmpSet oid="${oid}" community="${community}"
    version="${version}">{value}</snmpSet>
</request>
```

Below shows another SNMP set example with parameters:

```
<request>
  <snmpSet oid='${".1.3.6.1.2.1.1.4." + index}'
    community="public" version="2">{value}</snmpSet>
</request>
```

6.9.2 Response

This is where the response is converted to a result document.

The following is an example when using SNMP:

```
<request>
  <snmpGet oid=".1.3.6.1.2.1.1.4.0" community="public"
    version="2" />
</request>
<response>
  <bindings>
    <val name="value">
      <match>(.*)</match>
    </val>
  </bindings>
  <result xmlns:any="http://any.namespace.com/any/
    namespace/">
    <any:value>{value}</any:value>
```



```
    </result>
</response>
```

6.9.3 Runtime Parameters

Runtime parameters are used to store data from one command for use in a subsequent command in the same operation. For more information regarding Runtime parameters, see Section 9.5 on page 37.

6.10 HTTP Commands

The following command structure defines what is sent to the device when using HTTP:

```
<command>
  <request>
    (<httpGet>...</httpGet> or
     <httpPost>...</httpPost>)
  </request>
</command>
```

6.10.1 Request

Below shows an HTTP get example:

```
<request>
  <httpGet path="/a/b/c"/>
</request>
```

Below shows an HTTP POST example with parameters:

```
<request>
  <httpPost path="/a/b/c">test
    <messageBody>{value}</messageBody>
  </httpPost>
</request>
```

Below shows an HTTP POST example with XML content, and headers with parameters:

```
<request>
  <httpPost path="/a/b/c">
    <header name="Content-Type">
      text/xml; charset=ISO-8859-1
    </header>
    <header name="User Agent">{agent}</header>
    <messageBody>
      <xyz>{value}</xyz>
    </messageBody>
  </httpPost>
</request>
```




```

        </httpPost>
    </request>

```

6.10.2 Response

This is where the response is converted to a result document.

The following is an example when using HTTP:

```

<request>
  <httpPost path="/a/b/c">
    <header name="Content-Type">
      text/xml; charset=ISO-8859-1
    </header>
    <header name="User Agent">{agent}</header>
    <messageBody>
      <xyz>{value}</xyz>
    </messageBody>
  </httpPost>
</request>

<response>
  <bindings>
    <val name="value1">
      <xpath>header[@name='Content-Type']</xpath>
    </val>
    <val name="value2">
      <xpath>messageBody/xyz</xpath>
    </val>
  </bindings>
  <result xmlns:any="http://any.namespace.com/any/namespace/">
    <any:value1>{value1}</any:value1>
    <any:value2>{value2}</any:value2>
  </result>
</response>

```

6.11 Repeat

Repeat feature is used to create loops for iterating over lists. By specifying the attribute `forEachItemInList`, the repeat will loop as many times as there are items in the list. The attribute `bindEachItemTo` declares a variable to access the current item in the loop. The contents of a repeat defines a sequence of commands to be executed for each iteration.

If the list is empty, the repeat will not be performed.



Table 5 Repeat Definitions

Attribute	Cardinality	Description
forEachItemInList	Mandatory	Specifies the list to iterate over.
bindEachItemTo	Optional	Automatically create a binding, with the specified name, for the current item in the list.
bindings	Optional	Bindings at this level are local for the current iteration. For more information regarding bindings, see Section 6.6 on page 20.
command	0 to many	The sequence of commands to be executed to configure the feature. For more information regarding commands, see Section 6.7 on page 22. At least one of <code>command</code> or <code>repeat</code> is required.
repeat	0 to many	Defines a sequence of commands to execute for each item in a provided list. At least one of <code>command</code> or <code>repeat</code> is required.

6.12 Definitions

The Definitions can be used to predefine text variables in the template, for example, a Link definition file name.

```
<template>
...
<prompt/>
<definitions>
  <definition>DefinitionName</definition>
</definitions>
<globals/>
...
</template>
```



7 Vendor Template Definitions

A Definition is used to store reusable text assets. These can be referenced and used in Templates.

7.1 Definition Header

The Definition header contains name and description of the Definition.

```
<definition xmlns="http://schemas.ericsson.com/ma/definition/">
  <name>DEFINITION1</name>
  <description>
    A profound and descriptive text goes here.
  </description>
  ...
</definition>
```

7.2 Definition Bindings

Definition bindings are used to define variables to be used throughout templates. These bindings are provided to simplify the templates and reduce repetitiveness.

The difference between a definitions binding and a template binding is that definitions only support the usage of the def type.

7.2.1 Def

The following is an example of def type in a definition binding.

```
<bindings>
  <val name="localVariable">
    <def>"10"</def>
  </val>
</bindings>
```





8 Vendor Template Transformers

A Transformer is used to define reusable text transformations. These can be referenced and used in YANG Templates.

8.1 Transformer Header

The Transformer header contains name and description of the Transformer.

```
<transformer xmlns="http://schemas.ericsson.com/ma/transformer/">
  <name>TRANSFORMER1</name>
  <description>
    A profound and descriptive text goes here.
  </description>
  ...
```

8.2 Transformer Bindings

A transformation can be used to transform a value before it is sent to a device or to transform the response value from a device.

8.2.1 Function

The following is an example of function type in a transformer binding.

```
<bindings>
  <val name="appendWithText">
    <function value="value">
      "Append Text" + value
    </function>
  </val>
</bindings>
```

Table 6 Attributes in Transformer Binding

Attribute	Cardinality	Description
name	1	Name of transformation function visible in YANG Template.
value	1	Name of value sent into, and to be used by, the function.





9 Template Functions and Primitives

The template XML file contains several primitives that provide some generic functionality. This section is a list of the primitives and general functions available.

9.1 Naming Conventions

When defining new variables in parameter bindings, some key words are reserved for template syntax. These key words are specified below and must not be used as variable names.

abstract	false	lazy	Nil	true
case	final	match	return	type
catc	finally	new	sealed	throw
class	for	null	super	var
def	forSome	object	this	while
do	if	override	protected	with
else	implicit	package	trait	yield
extends	import	private	try	val

For example, `interfaceType` can be used as parameter name, but `type` is reserved and therefore must not be used.

9.2 Error Handling

9.2.1 Parameter Bindings

Parameter binding in the template can be used to resolve the xPath to an attribute value in the incoming request. It can also be used to provide default values when the attribute is missing in the inbound request.

The relations between bindings and parameters in the request is implicitly assuming that a request contains a value in the specified binding. Should this not be the case, either the parameter is missing or it is empty, the binding responds with an error.

To change this behavior, a default value is required in the binding. This default value specifies the value of the binding, if the binding fails to fetch data from the request.



```
eth:CreateEthernetInterface interfaceNumber="" interfaceType=
"FastEthernet">
    <eth:interfaceType>FastEthernet</eth:
interfaceType>
    <eth:interfaceNumber></eth:interfaceNumber>
    <!--Optional:-->
    <eth:duplex></eth:duplex>
    <eth:description></eth:description>
</eth:CreateEthernetInterface>

<bindings>
    <val name="interfaceType">
        <xpath>interfaceType</xpath>
    </val>
    <val name="interfaceNumberWithSub">
        <xpath>interfaceNumber</xpath>
    </val>
    <val name="duplex" defaultValue="full">
        <xpath>duplex</xpath>
    </val>
    <val name="description" defaultValue="">
        <xpath>description</xpath>
    </val>
</bindings>
```

Given the specified data above, the template bindings will give the following results:

```
interfaceType = "FastEthernet"
interfaceNumber = error
duplex = "full"
description=""
```

9.3 Parsing Output

9.3.1 Matching

There are three tags for matching and extracting parts from text, `<regex>`, `<match>`, and `<matchAll>`.

The `<regex>` and `<match>` tags are both using regular expressions to generate a single result for the response, whereas `<matchAll>` generates a list of results.

If there is no match for the regular expression in the different tags, the results are the following:



- In a `<regex>` tag the result will be an error message.
- In a `<match>` tag the result will be an empty string.
- In a `<matchAll>` tag the result will be an empty list.

9.4 Current Configuration

Configuration data is available since in certain circumstances it is not possible to successfully execute the commands without supplying additional information. In other circumstances, data can be lost after a command has been executed and that data needs to be reapplied.

Note: Current configuration is applicable for SET, GET and DELETE operations. During a CREATE or GETALL, there is no configuration in the repository.

The template always has access to the latest configuration from the repository during an operation. This data is accessible by specifying the source attribute with value **deviceConfiguration** on `<val>` tags in the binding section. Example is shown below:

```
<bindings>
  <val name="oldId" source="deviceConfiguration">
    <xpath>id</xpath>
  </val>
  <val name="newId">
    <xpath>id</xpath>
  </val>
</bindings>
```

9.5 Runtime Parameters

Runtime parameters are used to store data from one command for use in a subsequent command in the same operation. The runtime parameters are persistent over an entire operation, providing access to previous response data in subsequent commands.

For example, a runtime parameter defined in the `before` section can be used in a later command in the `before` section. The parameter can also be used during execution of the feature or even in the `after` section. In the same way, a runtime parameter defined during the execution of a feature can be used in the `after` section. However, this constructs a dependency in the `after` section for all features to define this parameter.



9.5.1 Save Runtime Parameter

Runtime parameters are defined as any other binding.

```
<command>
  <request>some output request</request>
  <runtimeParameters>
    <val name="param">
      <match>text (\d*) to match</match>
    </val>
  </runtimeParameters>
</command>
```

9.5.2 Use Runtime Parameter

To get the previously stored Runtime parameter, use `getRuntimeParameter(<nameOfVal>)`.

```
<command>
  <request>
    use previous ${getRuntimeParameter("param")}
    to make request
  </request>
</command>
```

10 Importing Devices

A typical Dynamic Activation solution in the wireless domain is handling a limited number of network elements that contain all information about the users in the network. Since the number of network elements is limited, they are all defined manually in the GUI.

When moving into broadband activation it is necessary to interact with many network elements and it is no longer possible to treat them as traditional Network Elements (NEs) in Dynamic Activation, setting them up one by one in the GUI. Instead, it is possible to import information about many devices in the network through a Comma-Separated-Value (CSV) file that typically could be generated by an inventory system.

An example of a device import is included in the delivery of Resource Configuration. To get the device import example, follow below instruction:

1. Save the zip file, [Resource_Configuration_examples.zip](#) to a temporary location.
2. Unpack the zip file.

When unzipping, there are two folders: `Templates` and `Device Import`.

3. Get the example from the `Device Import` folder and add it in a suitable location.

This example file describes how the device `mibf113` is imported.

10.1 Import Device JSON Format

Table 7 shows the format of the file used for import and export of devices. For more information regarding the allowed values, refer to *User Guide for Resource Configuration*, Reference [2].

Table 7 Data Attributes

Property	Type	Occurrence		Description
		POST	PUT	
masterIdentifier	String	M	N/A	The unique identifier of the device (not possible to update).
identifiers	Arrays	O	O	Additional unique identifiers of each device.



Property	Type	Occurrence		Description
		POST	PUT	
adminState	Array	M	M	Specifies the availability of the device, if set to the ACTIVE it is possible to provision the device otherwise the device cannot be provisioned. This parameter can be in OPERATION FAILED if an operation failed, for example a restore, then the user has to set it back to ACTIVE manually once the problem has been resolved.
modes	Array	O	O	Device modes.
CONFIGURATION_VALIDATION	-	O	O	Specifies whether the Configuration validation is enabled or not for the specific device and what action to take if there is an inconsistency between configuration. If enabled, a check is made that the actual configuration on the device matches the configuration last provisioned to the device. It must also be specified what action to take if the configurations do not match.
CANDIDATE_STORE	-	O	O	Specifies whether the Candidate store function is enabled or not for the specific device. If Candidate Store is enabled, the configuration for the selected device is stored in Candidate store before sent out to the device.
subModes	Array	O	O	Sub-modes property is only applicable for CONFIGURATION_VALIDATION. The following operations are available: <ul style="list-style-type: none"> • DISPLAY ERROR: The user will be notified with details of the discrepancies between the configurations. • UPDATE DEVICE: Dynamic Activation will update the device with the stored device configuration. • UPDATE REPOSITORY): Dynamic Activation will update the Resource Configuration device repository with the device configuration. If the configurations match, the device is provisioned with the new or updated configuration. If the configurations do not match, the action that was previously selected will occur.



Property	Type	Occurrence		Description
		POST	PUT	
types	Array	O	O	Specifies what type the device belong to. A device can belong to many types. This information is also used to decide which template to use when generating the southbound commands to the device.
accessPoint	Array	M	M	Access points for the device.
protocol	String	M	M	Specifies what protocol to use for communication with the device. Available protocols are SSH_CLI, SSH_NETCONF, and SNMP.
addresses	String	M	M	The address to the device according to format <code><IP_address>:<port></code> , for example 10.0.0.0:22.
parameters	key/value	O	O	Specifies a list of key value objects to set in the device. For example, providing a default parameter value.
credentials	key/value	M	M	Specifies the authentication credentials to use for communication with the device. These are provided as key value objects. The credentials must be recognized by the device and the keys "username" and "password" must be present. Optional Additional Parameter - Name can be added. For example, providing a default parameter value.
info	key/value	O	O	This is a field used by the system to inform about incidents, for example a restore that failed. The field is cleared if Admin state is changed from OPERATION FAILED to ACTIVE.
description	String	O	O	Free text where to enter additional information about the specific device.
category	String	O	O	Specifies what category the device belongs to.

10.2 Import Device CSV Format (Deprecated)

Note: Importing through CSV format is deprecated.

Table 8 shows the format of the file used for import and export of devices. Type `Multi value` specifies that the field can contain multiple values of



the content. The values are separated by using the “|” (pipe) character, for example, `deviceParameterName1|deviceParameterName2`. For more information regarding the allowed values, refer to *User Guide for Resource Configuration*, Reference [2].

Table 8 Import Device Format

Position	Content	Type	Allowed values	Restrictions
1	Master Identifier	Single value	Any string	Must not be empty
2	Admin State	Single value	ACTIVE, IDLE, UNKNOWN, INSTALLED, NOT INSTALLED, INACTIVE	Empty value will be set to UNKNOWN
3	Description	Single value	Any string	-
4	Additional identifiers		Any string	-
5	Device Category	Single value	CE, PE, FIREWALL, OTHER	-
6	Types	Multi value	Must match an id of a Template	-
7	modes	Multi value	CANDIDATE STORE, CONFIGURATION VALIDATION, FEATURE INSTANCE LOCK	-
8	configuration validation and feature instance lock, sub modes	Multi value	DISPLAY ERROR, UPDATE DEVICE, UPDATE REPOSITORY, max_device_connections=x where x could be between 2-100	-



Position	Content	Type	Allowed values	Restrictions
9	device information names	Multi value	Any string	#9 and # 10 must have the same number of values (each device information name must have a information value)
10	device information values	Multi value	Any string	
11	device credential names	Multi value	Any string	#11 and # 12 must have the same number of values (each device credential name must have a credential value)
12	device credential values	Multi value	Any string	
13	device parameter names	Multi value	Any string	#13 and # 14 must have the same number of values (each device parameter name must have a parameter value)
14	device parameter values	Multi value	Any string	
15	Address	Single value	<IPv4 host>:<port> or [<IPv6 host>]:<port>	-
16	protocol	Single value	SSH_CLI, SSH_NETCONF, SNMP, HTTP	-
17	additional parameter name	Multi value	Any string	#17 and # 18 must have the same number of values (each additional parameter name must have a parameter value)
18	additional parameter values	Multi value	Any string	





11 Configuring HTTPS Access Point

When adding a device with access point protocol HTTPS in the GUI, the default behavior is to trust the given device.

11.1 Additional Parameters

Parameters can be added to improve security with trust store and bi-directional authentication.

Table 9 Additional Parameters HTTPS

Parameters	Description
Trust store file	Path to trust store file to use. The trust store file must be: <ul style="list-style-type: none">• Created in PKCS#12 format.• Reachable from all nodes.
Trust store file password	Trust store file password.
Mutual authentication	When set to <code>true</code> , bi-directional authentication is supported, and key store must be defined.
Key store file	Path to key store file to use. The file must be reachable from all nodes.
Key store password	Key store password.





Reference List

Ericsson Documents

- [1] *Library Overview*, 18/1553-CSH 109 628 Uen
- [2] *User Guide for Resource Configuration*, 11/1553-CSH 109 628 Uen
- [3] *User Guide for Designer Studio*, 10/1553-CSH 109 628 Uen
- [4] *Provisioning over CAI3G for Resource Configuration*, 31/155 19-CSH 109 628 Uen