

# HP Service Activator

## TeMIP Liaison

**Edition: V51-1A**



**Manufacturing Part Number: None**

**July 1, 2010**

© Copyright 2010 Hewlett-Packard Development Company, L.P.

## Legal Notices

### Warranty.

*Hewlett-Packard makes no warranty of any kind with regard to this manual, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.*

A copy of the specific warranty terms applicable to your Hewlett-Packard product can be obtained from your local Sales and Service Office.

### Restricted Rights Legend.

Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause in DFARS 252.227-7013.

Hewlett-Packard Company  
United States of America

Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19(c)(1,2).

### Copyright Notices.

©Copyright 2001-2010 Hewlett-Packard Development Company, L.P., all rights reserved.

No part of this document may be copied, reproduced, or translated to another language without the prior written consent of Hewlett-Packard Company. The information contained in this material is subject to change without notice.

### Trademark Notices.

Java™ is a U.S. trademark of Sun Microsystems, Inc.

Linux is a U.S. registered trademark of Linus Torvalds

Microsoft® is a U.S. registered trademark of Microsoft Corporation.

MS-DOS® is a U.S. registered trademark of Microsoft Corporation.

Oracle® is a registered U.S. trademark of Oracle Corporation, Redwood City, California.

UNIX® is a registered trademark of the Open Group.

Windows® and MS Windows® are U.S. registered trademarks of Microsoft Corporation.

All other product names are the property of their respective trademark or service mark holders and are hereby acknowledged.

Document id: p158-pd010004

# Table of Contents

<b>Chapter 1: Introduction.....</b>	<b>8</b>
Calling TeMIP directives from HP Service Activator.....	8
Running OVSA workflows from TeMIP.....	9
<b>Chapter 2: Configuring <i>TeMIP liaison</i> to work.....</b>	<b>10</b>
Overview.....	10
Configuring Service Activator side.....	11
TeMIP module configuration.....	11
TeMIPLiaison plug-in configuration.....	11
Installing and configuring TeMIP side.....	12
Installation.....	12
Configuration.....	13
Backup of MIR.....	14
Re-start of TeMIP OVSA FM.....	14
<b>Chapter 3: TeMIP Calls and Results on Service Activator .....</b>	<b>16</b>
Call Syntax.....	16
TeMIP Entity Names.....	17
Argument and Attribute Values.....	17
TeMIP Result Syntax.....	19
Examples.....	20
<b>Chapter 4: OVSA FM .....</b>	<b>24</b>
Overview.....	24
Service interface.....	24
OVSA.....	24
Queue.....	26
Job.....	27
Workflow.....	30
Self-management interface.....	31
OVSA.....	31
OVSA directives.....	31
OVSA config class.....	32
<b>Chapter 5: TeMIP nodes.....</b>	<b>34</b>
Overview.....	34
TeMIPExecuteDirective Examples.....	35
Non wild-carded Show Call.....	35
Wild-carded Show Call.....	37
Action and modification calls.....	38
TeMIPStartDirective and TeMIPNextResult Example.....	41
TCL execution.....	43

Variable substitution.....	44
<b>Chapter 6: TeMIPLiaison plug-in.....</b>	<b>46</b>
Overview .....	46
Transaction rollback .....	46
Temp_EXP_directive Example .....	47
Variable substitution.....	47
Template files .....	48

## Install Location Descriptors

The following names are used to define install locations throughout this guide.

Descriptor	What the Descriptor Represents
<i>\$ACTIVATOR_OPT</i>	The base install location of Service Activator. The UNIX® location is /opt/OV/ServiceActivator The Windows® location is <install drive>:\HP\OpenView\ServiceActivator
<i>\$ACTIVATOR_ETC</i>	The install location of specific Service Activator files. The UNIX location is /etc/opt/OV/ServiceActivator The Windows location is <install drive>:\HP\OpenView\ServiceActivator\etc
<i>\$ACTIVATOR_VAR</i>	The install location of specific Service Activator files. The UNIX location is /var/opt/OV/ServiceActivator The Windows location is <install drive>:\HP\OpenView\ServiceActivator\var
<i>\$ACTIVATOR_BIN</i>	The install location of specific Service Activator files. The UNIX location is /opt/OV/ServiceActivator/bin The Windows location is <install drive>:\HP\OpenView\ServiceActivator\bin
<i>\$ACTIVATOR_THIRD_PARTY</i>	The location for new Java™ components such as workflow nodes and modules. The UNIX location is /opt/OV/ServiceActivator/3rd-party The Windows location is <install drive>:\HP\OpenView\Service Activator\3rd-party Customized inventory files are stored in the following locations: UNIX: <i>\$ACTIVATOR_THIRD_PARTY</i> /inventory Windows: <i>\$ACTIVATOR_THIRD_PARTY</i> \inventory
<i>\$JBOSS_HOME</i>	The install location for JBoss. The UNIX location is /opt/HP/jboss The Windows location is <install drive>:\HP\jboss
<i>\$JBOSS_DEPLOY</i>	The install location of the Service Activator J2EE components. The UNIX location is /opt/HP/jboss/server/default/deploy The Windows location is <install drive>:\HP\jboss\server\default\deploy
<i>\$ACTIVATOR_DB_USER</i>	The database user name you define. Suggestion: ovactivator
<i>\$ACTIVATOR_SSH_USER</i>	The Secure Shell user name you define. Suggestion: ovactusr

## In This Guide

This guide contains information about HP Service Activator TeMIP liaison solution which is provided as part of HP Service Activator.

The guide contains detailed information about:

- Installation and configuration of TeMIP liaison
- Syntax of command messages and responses to them
- The syntax of encoding of values of different TeMIP data types
- The recommendations and examples of usage of TeMIP workflow nodes
- The recommendations and examples of usage of TeMIPLiaison plug-in

## Audience

The audience for this guide is:

- Systems Integrator, using it as a resource for building new solutions.
- Educational staff, using it as student material in customer training.

The reader must understand the architecture, tools, and service delivery processes described in *HP OpenView Service Activator–Introduction & Overview* Guide.

In addition, the reader has a combination of some or all of the following:

- Understands the XML and DTD schemes
- Has a basic knowledge of TCL language and TTS commands
- Has a basic understanding of TeMIP framework and especially of following areas:
  - Data types
  - Dictionary structure
  - Directive calls

## References

[MWFM] HP OpenView Service Activator, Workflows and the Micro-Workflow Manager.

[RESMGR] HP OpenView Service Activator, Developing plug-ins and compound tasks.

[TTS] HP OpenView TeMIP TCL Scripting User Guide.

# Manual Organization

This guide contains the following chapters:

Chapter 1, “Introduction”, which describes the structure of the *TeMIP liaison* solution, its features and possible areas of application.

Chapter 2, “Configuring *TeMIP liaison* to work”, which provides detailed instructions on how to install the the *TeMIP liaison* solution and configure it.

Chapter 3, “TeMIP data encoding”, which describes what request and response messages the communication of TeMIP and HP Service Activator is based on, how different TeMIP data types are encoded into XML.

Chapter 4, “OVSA FM”, describes the model of the central part of the *TeMIP liaison* solution – OVSA FM.

Chapter 5, “TeMIP nodes”. As well as the detailed descriptions of TeMIP nodes is provided in the document [MWFM], this chapter contains a recommendations and examples of usage of TeMIP nodes.

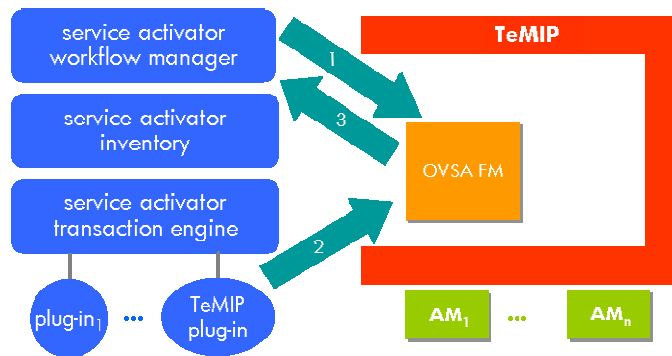
Chapter 6, “TeMIPLiaison plug-in”, provides the examples of usage of TeMIPLiaison plug-in.

# Chapter 1: Introduction

The HP Service Activator *TeMIP Liaison* (further *TeMIP liaison* or *liaison*) is intended to facilitate the building of solutions which draw on the combined power of the two platforms – HP TeMIP and HP Service Activator - and leverage existing solution components from both of them, components which may or may not have been productized. The liaison requires TeMIP V5.0 on HP-UX PA-RISC.

The liaison contains for each platform components which fit into its architecture and expose the native mechanisms for retrieving data and controlling activities on the other platform. The main mechanisms for interactions are depicted in Figure 1. Note that a special functional module, the OVSA FM, is introduced on the TeMIP platform and serves as a key component of the liaison.

**Figure 1, Liaison interaction mechanisms**



## Calling TeMIP directives from HP Service Activator

In order to enable an HP Service Activator solution to make use of capabilities implemented on TeMIP, the TeMIP directive call is made available for use within workflows. TeMIP directive calls can be invoked in two different ways that are suitable for different purposes. The first way is through workflow nodes that are added to the node library so that they can easily be used in workflows (1 in Figure 1). The second way is through a TeMIP-call plug-in (2 in Figure 1) which can be called from a workflow as an activation step. The plug-in can be used when an activation task is supported on TeMIP and then allows it to be invoked in the same way as an activation task in any other plug-in.

The workflow nodes are intended for retrieval of data that is available in the TeMIP information model, when such data is needed in the activation sequencing and parameter identification logic of an Service Activator workflow. Five nodes are provided. One, `TeMIPExecuteDirective`, takes parameters which fully specifies a TeMIP call, makes the call, waits for the execution of the call to complete, and returns all resulting data. If the call returns multiple responses, the node waits for all of them to be returned.

Three other nodes, `TeMIPStartDirective`, `TeMIPNextResult` and `TeMIPCancelDirective`, are for asynchronous response processing. They allow the separation of call initiation and result retrieval, with the possibility to cancel an active call. When a call returns multiple results, they are retrieved one at a time. To simplify branching on normal responses and failure cases `TeMIPNextResult` is a rule node.

Arguments of the TeMIP directive call are given by a single parameter, XML encoded to hold the names and values of all relevant arguments. Similarly results are returned in XML encoded form which is easily parsed by the standard features of Service Activator.

Finally, the fifth node, `TeMIPExecTcl`, supports the execution of a TCL command which can draw on the features of TTS, TeMIP Tcl Scripting. The command is passed as a parameter to the node and is



executed by the OVSA FM. Results are returned through an output parameter of the node. This node makes it possible to retrieve TeMIP information which cannot be obtained by invoking a directive, for example from the TeMIP dictionary. For more information on the power of TTS, please refer to TTS documentation.

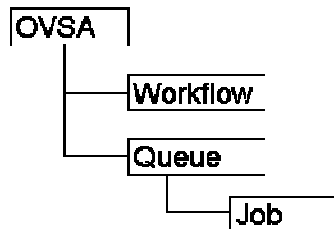
All activity is initiated by messages forwarded from the nodes and plug-in described above and received by the OVSA FM on the TeMIP side. The FM will then make the implied call or calls into the TeMIP framework.

A general message exchange protocol, not visible to the system integrator, enables the Service Activator and TeMIP components to communicate as described above.

## Running OVSA workflows from TeMIP

OVSA supports a general API for controlling its activity and inspecting its state. The native API is a Java RMI. A SOAP version is used within the TeMIP Service Activator Liaison to start workflow jobs, inspect them when queued and interacting with them (3 in Figure 1). The OVSA FM, using the SOAP interface, exposes for use by other TeMIP modules the TeMIP framework the information model shown in Figure 2:

**Figure 2, TeMIP model of OVSA**



An instance of the class OVSA represents a Service Activator server. These are the only entities persisted on TeMIP. All child entities are proxies for objects on the Service Activator server.

A Workflow entity represents a workflow that is available on the parent server. The list of workflows available on a server can be retrieved to TeMIP by a wild-carded Show call.

A job running a specific workflow can be started by the Start directive (on class Workflow), which will return the job id. A single argument on this directive will be passed to the job and can be used to initialize case-packet variables. How to encode multiple variable values within the argument is a convention between the caller and the workflow; the general recommendation is to use an XML-encoded list of names and values, which will be easy to decode within the workflow.

A Queue entity represents an Service Activator queue. Two queues are always present: 'Running Jobs' and 'Scheduled Jobs'. All current queues can be listed by a wildcarded Show call.

A Job entity represents a Service Activator job which is waiting on the queue included in the full entity name. A job can be killed by means of the Stop directive. Synchronization with the job, providing it with the data it is waiting for, can be achieved by means of the Load directive, which will load values provided as through a directive argument structured as a list of (name, value) pairs into case-packet variables of the workflow job, just as if an operator had interacted with the job on Service Activator's GUI.

## Chapter 2: Configuring *TeMIP liaison* to work

This chapter describes how to install and configure the entire *TeMIP liaison* solution.

### Overview

The TeMIP OVSA Liaison solution, as mentioned in previous chapter, consists of two sides. The first one – OVSA side – represented by the MWFM module TeMIPModule and TeMIPLiaison plug-in is built into Service Activator. It does not require additional installation – it should only be configured for communication with one or a number of TeMIP directors.

The TeMIP side of the *TeMIP liaison* consists of one component, the OVSA FM. It should be installed and configured to make *TeMIP Liaison* solution functioning.

In the following sections you can find information about how to install and configure the *TeMIP liaison*. To make a description of the installation and configuration process more illustrative an example situation is used. Let's assume that:

- The communication of single Service Activator instance to single TeMIP director should be organized
- your Service Activator installation and the OVSA FM are running on different machines:
- the Service Activator instance is running on the host **172.16.1.28**
- the OVSA FM is running on the host **172.16.1.35**
- the OVSA FM is listening to the requests from Service Activator instances on port number **4051**
- the TeMIPLiaison plug-in is listening to the OVSA FM on port **3074**
- the TeMIPModule is listening to the OVSA FM on port **3075**
- the ID of the connection of TeMIPLiaison plug-in to the OVSA FM is **RESMGR\_sa0\_director0**,
- the ID of the connection of TeMIPModule to the OVSA FM is **MWFM\_sa0\_director0**
- The user id, which is used when sending requests from the OVSA FM to the Service Activator's web/soap interface, is **example\_usr**
- The password, which is used when sending requests from the OVSA FM to the Service Activator's web/soap interface, is **example\_pwd**
- the port number where the web-service is listening for requests from the OVSA FM is **8080**

## Configuring Service Activator side

This section explains how to configure the Service Activator side of the *TeMIP liaison*.

### TeMIP module configuration

The configuration of all of the MWFM modules is placed into file `$ACTIVATOR_ETC/config/mwfm.xml`.

For the description of the configuration parameters of the TeMIPModule see section TBD in TBD.

To configure TeMIPModule do the following:

- in `mwfm.xml` find the line “Start TeMIP module” and uncomment all the lines below it until “End TeMIP module”. As the result you will get the following lines uncommented:

```
<Module>
  <Name>temip</Name>
  <Class-Name>
    com.hp.ov.activator.mwfm.engine.module.TeMIPModule
  </Class-Name>
  <Param name="localport" value="3073" />
  <Param name="director0" value="MWFM 172.16.0.29 4050" />
  <Param name="confirmtime" value="4000" />
  <Param name="polltime" value="30" />
  <Param name="defaulttimeout" value="30000" />
  <Param name="keepaliveinterval" value="5000" />
  <Param name="keepalivetimeout" value="60000" />
</Module>
```
- According to the example situation change the module parameters as follows:
- Set the value of the parameter *localport* to **3075**
- Set the value of the parameter *director0* to “**MWFM\_sa0\_director0 172.16.1.35 4051**”
- Set parameters *confirmtime*, *polltime*, *defaulttimeout*, *keepaliveinterval*, *keepalivetimeout* to your preferred values (description of these parameters can be found in [MWFM], section TBD)

### TeMIPLiaison plug-in configuration

To use the TeMIPLiaison plug-in the plug-in must be deployed.

The TeMIPLiaison plug-in configuration settings should be placed into file `$ACTIVATOR_ETC/config/temipResources.xml`. The content of this file conforms to the DTD file `temipResources.dtd` placed in the same directory.

As far as `temipResources.xml` is not provided with the Service Activator installation (only the `temipResources.dtd`), it should be created explicitly with the below content to configure plug-in according to example parameters:

```
<TeMIPResources>
  <ListenerPort>3074</ListenerPort>
  <DefaultTimeout>60000</DefaultTimeout>
  <KeepAliveInterval>5000</KeepAliveInterval>
```

```

<KeepAliveTimeout>60000</KeepAliveTimeout>
<PollTime>30</PollTime>
<ConfirmTime>30000</ConfirmTime>
<Connection>
  <CID>RMGR_sa0_director0</CID>
  <TeMIPHost>172.16.1.35</TeMIPHost>
  <TeMIPPort>4051</TeMIPPort>
</Connection>
</TeMIPResources>

```

The correspondence between elements in `temipResources.xml` and parameters of `TeMIPModule` in `mwfm.xml` is as follows (in the context of `TeMIPLiaison` plug-in, of course):

- `ListenerPort` = `localport` – the port the *TeMIP liaison* listens to the messages directed to the plug-in
- `DefaultTimeout` = `defaulttimeout`
- `KeepAliveInterval` = `keepaliveinterval`
- `KeepAliveTimeout` = `keepalivetimeout`
- `PollTime` = `polltime`
- `ConfirmTime` = `confirmtime`
- `Connection` = `directorN` – describes the parameters of the connection of the `TeMIPLiaison` plug-in to `TeMIP` director. Sub-element `CID` corresponds to the first token in the value of the parameter `directorN` (`connectionId`), `TeMIPHost` corresponds to the second one (`temiphost`) and `TeMIPPort` to the third token (`temipport`). There can be a number of elements `Connection`.
- The purpose of each of the elements can be found in TBD.
- Set elements *DefaultTimeout*, *KeepAliveInterval*, *KeepAliveTimeout*, *PollTime* in `temipResources.xml` to the values you intend to use in your solution for the connections of the plug-in to `TeMIP` directors (description of these parameters can be found in [RESMGR], section TBD). These values can differ from analogous set in `mwfm.xml` in configuration of `TeMIPModule`.

## Installing and configuring TeMIP side

The section below describes how to install and configure the `TeMIP` side of the *TeMIP liaison*, i.e. the `OVSA FM`.

The `TeMIP` side is fully based on the `TTS` product [TTS] version 5.0. Before starting the installation assure that `TTS` is installed on the `TeMIP` director. If `TTS` is not installed then follow the instructions described in [TTS]. The `TTS` kit and documentation can be found in the `Unix/TeMIPLiaison/TTS` directory on the `Service Activator` CD. Likewise can a license file be found in this directory. Please refer to the `TeMIP` documentation about how to install a license.

### Installation

The `OVSA FM` is not automatically installed when `Service Activator` is installed. A separate kit exists, which can be found in the `unix/OVACTFM/OVDEPOT_HPUX11.11` directory on the `Service Activator` CD.

To install the `OVSA FM` do:

```
swinstall -s <kit> OVACTFM:<temip>
```

Where <kit> is the absolute path to the directory where the kit can be found and <temip> is the absolute path to the active temip version (normally /usr/opt/TEMIPV500)

To uninstall the OVSA FM do:

```
swremove OVACTFM,l=<temip>
```

## Configuration

The procedure of configuration of TeMIP side consists of two steps:

- The configuration of the global parameters of the OVSA FM. These parameters are common for all OVSA instances of the FM and are initialized at the OVSA FM start-up. They are the attributes of the class “config”, which is a subclass of OVSA FM self-management interface. The attributes are :
  - portNumber – the number of the port, on which the OVSA FM listens to the connections from all of Service Activator instances
  - connConfirmTimeout – the time in milliseconds between attempts of OVSA FM to resume connection to Service Activator instance if it has been lost.
  - confirmTimeout – the time in milliseconds within which the request sent by OVSA FM to one of Service Activator instances should be confirmed.
- Creation of instances of the global class OVSA. Each instance represent the configuration parameters for setting up the communication between the TeMIP director and one Service Activator installation.

Following the configuration of TeMIPModule and TeMIPLiaison plug-in, which have been set in the previous sections, the below steps should be undertaken on TeMIP side to accomplish the *TeMIP liaison* configuration and make it functioning.

Using the *tts\_pm* or *manage set* the attributes of subclass “config” of the OVSA FM self-management interface to the values you intend to use in your solution. See section “When a configuration parameter is set it does not take effect until OVSA FM is restarted.

- “OVSA config class attributes” for details about attributes of class “config”. E.g. you intend to set portNumber to 4050, the connConfirmTimeout to 10000 and confirmTimeout to 10000. In manage the following command should be executed:

```
TeMIP> set mcc 0 ovsa config portNumber 4050, -  
_TeMIP> connConfirmTimeout 10000, -  
_TeMIP> confirmTimeout 10000
```

Using the *tts\_pm* or *manage create* OVSA instance with the following arguments set for the directive call:

```
hostName = 172.16.1.28  
portNumberMWFm = 3075  
portNumberRESMGR = 3074  
portNumberWEB = 8080  
userId = example_usr  
password = example_pwd  
cidMWFm = MWFm_sa0_director0  
cidRESMGR = RESMGR_sa0_director0  
description = Test instance.
```

In manage the following command should be executed:

```
TeMIP> create ovsa sa0 -
_TeMIP> hostname 172.16.1.28, -
_TeMIP> portNumberMWFM 3075, -
_TeMIP> portNumberRESMGR 3074, -
_TeMIP> portNumberWEB 8080, -
_TeMIP> userId example_usr, -
_TeMIP> password example_pwd, -
_TeMIP> cidMWFM MWFM_sa0_director0, -
_TeMIP> cidRESMGR RESMGR_sa0_director0, -
_TeMIP> description "Test instance"
```

Re-start OVSA FM (see section “Re-start of TeMIP OVSA FM”). The restart is required only in the case if the configuration parameters of OVSA FM self-management interface have been changed. If only an OVSA instance is created (for example, if you have TeMIP Liaison already running but would like to add another OVSA instance) the restart is not required – OVSA FM will automatically reconfigure to also communicate to this instance.

---

## NOTE

During the installation of TeMIP side the application entity associated with the OVSA FM (MCC 0 APPLICATION ovsa) is created and its attribute “Automatic Startup” is set to true. It means that OVSA FM will be automatically started when TeMIP is started. However if the OVSA FM is manually stopped then no request from the OVSA side will be executed. The OVSA FM must be running to process these requests.

**The OVSA FM must be running all the time to have *TeMIP Liaison* fully functioning.**

---

## Backup of MIR

The configuration of self-management interface (see section “When a configuration parameter is set it does not take effect until OVSA FM is restarted.

OVSA config class attributes”) as well as instances of class OVSA (see section “OVSA”) are persisted in a local MIR and are loaded into the memory each time the OVSA FM is started. The MIR is placed in the directory /var/opt/temip/ovsa.

You can populate your OVSA FM installation with existing, already configured OVSA instances. Simply place your file config.dat into the directory /var/opt/temip/ovsa/.

On the other hand if you intend to reinstall OVSA FM then make a back-up copy of this file.

## Re-start of TeMIP OVSA FM

The OVSA FM is represented by the “mcc 0 application ovsa” instance. If you want to restart the FM then you must first stop the FM and then start the FM. The stop and start directive are found on the “application” class. In manage a restart of the FM must be executed the following way:

```
TeMIP> stop mcc 0 application ovsa
```

```
TeMIP> start mcc 0 application ovsa
```



## Chapter 3: TeMIP Calls and Results on Service Activator

This chapter explains the XML syntax used to make TeMIP calls from Service Activator and decode the results.

The syntax is basically the same whether a TeMIP call is made using workflow nodes or the plug-in. The XML syntax covers a complete description of a TeMIP call or result from a call. With the workflow nodes it is also possible to pass the various parts of the call description (entity, verb, arguments) as node parameters. When this is done only the arguments need to be encoded in XML. When the entire call is described in XML the description may be held in a file or in a node parameter. Refer to [ ] for documentation of node parameters.

Results are always XML wrapped and must be parsed by the receiving workflow.

From a workflow a call which will return multiple results can be made in two ways: using the ExecuteDirective node to return all the results lumped together, or using the StartDirective node to return one result at a time. Only the first option is available in the plug-in.

Before the formal syntax definition here is an example ExecuteDirective to give the flavor. The call sets a value for the attribute “Trace File” of the instance “mcc 0 application tts\_pm”:

```
<ExecuteDirective>
  <User>Hugo</User>
  <Expression>
    <Verb>set</Verb>
    <Entity>
      {mcc 0} {application tts_pm}
    </Entity>
    <Partition>Characteristics</Partition>
    <Arguments>
      <Attributes>
        <Trace__File tempName="Trace File">
          <Value>/tmp/trace_file.log</Value>
        </Trace__File>
      </Attributes>
    </Arguments>
  </Expression>
</ExecuteDirective>
```

The following sections describe call and result syntax, respectively, and also briefly describe the meaning of the syntactic elements.

### Call Syntax

The XML syntax to describe a TeMIP call is identical in the two cases, except for the tag. The DTD syntax for ExecuteDirective and StartDirective, respectively, is as follows:

```
<!ELEMENT ExecuteDirective (User?, Expression)>
<!ELEMENT StartDirective (User?, Expression)>
```

The DTD for the identical bodies follow. Note that tags in italics are “meta-tags”, not to be taken literally, as explained in the subsection Argument and Attribute Values below.

```
<!ELEMENT User (#PCDATA)>
<!ELEMENT Expression (Vep | (Verb, Entity, Partition?), Qualifiers?,
Arguments?)>
<!ELEMENT Vep (#PCDATA)>
```



```

<!ELEMENT Verb (#PCDATA)>
<!ELEMENT Entity (#PCDATA)>
<!ELEMENT Partition (#PCDATA)>
<!ELEMENT Qualifiers (domain | password | account | user | manager)*>
<!ELEMENT domain (#PCDATA)>
<!ELEMENT password (#PCDATA)>
<!ELEMENT account (#PCDATA)>
<!ELEMENT user (#PCDATA)>
<!ELEMENT manager (#PCDATA)>
<!ELEMENT Arguments (Attributes?, Args*)>

```

The leaf elements `User`, `Verb`, `Partition`, `domain`, `password`, `account`, `user` and `manager` all have the form of simple strings.

If the `User` element is present an acloc session is created on the TeMIP system and the call is executed in this session. Otherwise it is executed in an acloc session belonging to the default user, `temip`.

The element `Entity` represents the in-entity of the call, see the subsection `TeMIP Entity Names` below.

The element `Vep` can be used to concatenate the values of `Verb`, `Entity` (enclosed within an extra pair of curly braces) and optionally `Partition`, into a single value, for example:

```
<Vep>set {{mcc 0} {application tts_pm}} Characteristics</Vep>
```

The element `Arguments` represents an optional list of argument or attribute values. It is explained in the subsection `Argument and Attribute Values` below.

## TeMIP Entity Names

TeMIP entity names appear as the element `Entity` in XML-coded calls and results, and they can also appear as argument or attribute values. They are list structured, but for simplicity the structure is not rendered using XML tags. Instead, the standard TeMIP FCL syntax is used with the addition of curly braces around each (class name, instance name) pair, where the instance name is optional, thereby potentially causing ambiguity when a name is parsed without the context of the TeMIP dictionary. Names containing spaces must be enclosed within extra curly brackets and then within double quotes (example: {"a b"}). As usual, the global class name can be preceded by a TNS name space name.

Here is an example which includes a TNS name space:

```
{domain mach5:.denmark}{member xsc17}
```

Wildcarding (full or partial) can be used in instance names, normally to retrieve (Show) data from multiple entities, resulting in multiple responses to the call.

Synonyms are not supported.

## Argument and Attribute Values

The element `Arguments` in the XML syntax for a call or result represents a list of call or result arguments. In a modify (Set) directive the `Attributes` element can be used within the `Arguments` element to contain a list of attribute (name, value) pairs:

```

<!ELEMENT Attributes (Atr*)>
<!ELEMENT Atr (Value)>
<!ATTLIST Atr temipName PCDATA #REQUIRED>

```

In action and getevent directives the `Arguments` element contains a list of `Arg` elements each describing one call argument:

```
<!ELEMENT Arg (Value)>
<!ATTLIST Arg tempName PCDATA #REQUIRED>
```

The element *Value* represents the value of a single attribute (or argument, see below). If the value is of a simple TeMIP type (numeric or string) it is contained directly in the *Value* element using standard TeMIP syntax for simple values:

```
<!ELEMENT Value (#PCDATA)>
```

The following simple TeMIP types are supported: Boolean (can be encoded as “yes”, “no”, “true”, “false”, “1”, “0”, “on”, “off”, decodes to “true” or “false”), OctetString, Latin1String, SimpleName, FullName, FileSpec, UID, Version, Enumeration, Expression, IntegerXX, UnsignedXX, CounterXX, LcounterXX, MCCError, Octet, HexString, BinAbsTime (CCYY-MM-DD-hh:mm:ss.ffff), BinRelTime, DictionarySpec, FloatF, Real, IPAddress, InternetName

If the value is an entity name or of a TeMIP constructor or constructed data type further XML encoding is used to convey its structure:

```
<!ELEMENT Value (Entity | Record | List | EventReport | AttribList)>
```

Definitions of the elements *Record*, *List*, *EventReport* and *AttribList* are given under individual headings below; *Entity* is discussed under the heading TeMIP Entity Names above.

In results from *Set* and *Show* directives the *Attributes* element is also used but in a slightly different form from the one specified above, as it contains for each value also a reason code:

```
<!ELEMENT Attributes (Atr*)>
<!ELEMENT Atr (Value?, ReasonCode)>
<!ATTLIST Atr tempName PCDATA #REQUIRED>
<!ELEMENT ReasonCode (#PCDATA)>
```

If *ReasonCode* is different from “Available”, the *Value* will not be present.

Similarly, a reply argument in the result from an action or *getevent* directive will use the following form of the *Arg* element, i.e. there will be a *ReasonCode*, explaining the possible absence of the *Value*:

```
<!ELEMENT Arg (Value?, ReasonCode)>
<!ATTLIST Arg tempName PCDATA #REQUIRED>
```

In all of the constructs discussed here the “meta-tag” of the element representing a single attribute or argument (*Atr* or *Arg*) represents the display name of the attribute or argument with each space replaced by two underscores. The actual TeMIP display name is given as the value of the *tempName* (XML) attribute within double quotes.

## Record

```
<!ELEMENT Record (Field*)
<!ELEMENT Field (Value)>
```

The *Field* element tag is the name of the field.

For example, a record with two fields named X and Y containing simple values will be represented as:

```
<Record>
  <X>12345</X>
  <Y>ABC</Y>
</Record>
```

## SetOf, SequenceOf, AttribIdList and EventIDList

The values of these four data types have a list structure and are encoded in the same way.

```
<!ELEMENT List (Element*)>
<!ELEMENT Element (Value)>
```

## EventReport

The substance of a getevent reply is normally an argument named “Event Data” of type EventReport. This type is similar to an argument list and reuses the principle for naming an element of the list and also the syntax elements Value and ReasonCode.

```
<!ELEMENT EventReport (EventId, EventArguments)>
<!ELEMENT EventId (#PCDATA)>
<!ELEMENT EventArguments (EventArg*)
<!ELEMENT EventArg (Value, ReasonCode)>
<!ATTLIST EventArg tempName PCDATA #REQUIRED>
```

## AttribList

The element tagged with the “meta-tag” *Atr* is as described above in the main section for its appearance in the Attributes element.

```
<!ELEMENT AttribList (Atr*)>
```

## TeMIP Result Syntax

The workflow node TeMIPNextResult retrieves one result from a TeMIP call made by means of the StartDirective node. If there are multiple results, they must be retrieved one at a time. The syntax for the retrieved result is the OK element which is described below.

The workflow node TeMIPExecuteDirective waits for all results from the specified call to be returned. The result syntax for retrieved results is:

```
<!ELEMENT TemipResults (Handle, ResponseNumber, OK*, ResultCount,
ErrorFlag)>
```

The DTD syntax for the elements which occur in the body of TeMIP results is:

```
<!ELEMENT Handle (#PCDATA)>
<!ELEMENT ResponseNumber (#PCDATA)>
<!ELEMENT OK (Time, Entity, ResultStatus, Director, Arguments?)>
<!ELEMENT Time (#PCDATA)>
<!ELEMENT ResultStatus (Code, Status, Text)>
<!ELEMENT Code (#PCDATA)>
<!ELEMENT Status (#PCDATA)>
<!ELEMENT Text (#PCDATA)>
<!ELEMENT Director (#PCDATA) >
<!ELEMENT More (#PCDATA) >
<!ELEMENT ResultCount (#PCDATA)>
<!ELEMENT ErrorFlag (#PCDATA)>
```

The element Arguments is discussed under Argument and Attribute Values in the Call Syntax section. All the other elements contain standard TeMIP call response information in standard encoding except the last two. ResultCount provides the number of results (number of OK elements) in the entire result, and ErrorFlag is a boolean indicating whether the entire result contains valid response information.

## Examples

An ExecuteDirective was given at the beginning of this chapter, setting a value for the attribute “Trace File” of the instance “mcc 0 application tts\_pm”. The result returned by the ExecuteDirective from the node or the plug-in would look like:

```
<TeMIPResults>
  <Handle>1111397617433#TeMIPStartDirective#-
552608615852567522</Handle>
  <ResponseNumber>1</ResponseNumber>
  <OK>
    <Time> Thu Oct 7 09:48:46</Time>
    <Entity>
      {osi_system} {testobj xxx}
    </Entity>
    <ResultStatus>
      <Code>1</Code>
      <Status>Response</Status>
      <Text>
        {Set Success} {Modification(s) completed successfully.}
      </Text>
    </ResultStatus>
    <Director>.:peony_ns</Director>
    <Arguments>
      <Attributes>
        <Trace__File temipName="Trace File">
          <Value>123456789</Value>
          <ReasonCode>Available</ReasonCode>
        </Trace__File>
      </Attributes>
    </Arguments>
  </OK>
  <ResultCount>1</ResultCount>
  <ErrorFlag>>false</ErrorFlag>
</TeMIPResults>
```

This StartDirective describes a call to show the attributes in the characteristics partition of domain testdomain.

```
<StartDirective>
  <User>Hugo</User>
  <Expression>
    <Verb>show</Verb>
    <Entity>
      {domain testdomain}
    </Entity>
    <Partition>Characteristics</Partition>
    <Qualifiers>
      <domain>Vilnius</domain>
      <user>Hugo</user>
    </Qualifiers>
  </Expression>
</StartDirective>
```

The TeMIPStartDirective node returns a handle to the TeMIP directive call that has been started. The handle can be used in the TeMIPNextResult node to retrieve the result. The result returned by the TeMIPNextResult will be the OK element would look like:

```
<OK>
  <Time>2005-03-21-10:42:39</Time>
  <Entity>{Domain lithium_ns:.testdomain}</Entity>
  <ResultStatus>
    <Code>1</Code>
    <Status>Response</Status>
    <Text>{Show Success} {Examination of attributes shows}</Text>
  </ResultStatus>
  <Director>lithium_ns:.temip.lithium_director</Director>
```

```

<Arguments>
  <Attributes>
    <Owner__ID tempName="Owner ID">
      <Value>rta</Value>
      <ReasonCode>Available</ReasonCode>
    </Owner__ID>
    <Directory tempName="Directory">
      <Value>/home/rta</Value>
      <ReasonCode>Available</ReasonCode>
    </Directory>
    <Domain__category tempName="Domain category">
      <Value></Value>
      <ReasonCode>NotAvailable</ReasonCode>
    </Domain__category>
  </Attributes>
</Arguments>
</OK>

```

An example of ExecuteDirective that uses wildcarding and returns multiple responses follows. It shows the characteristics attributes of all domains.

```

<ExecuteDirective>
  <User>Hugo</User>
  <Expression>
    <Verb>show</Verb>
    <Entity>
      {domain *}
    </Entity>
    <Partition>char</Partition>
  </Expression>
</ExecuteDirective>

```

The result could be the following:

```

<TeMIPResults>
  <Handle>1111397617433#TeMIPStartDirective#-
552608615852567522</Handle>
  <ResponseNumber>1</ResponseNumber>
  <OK>
    <Time>2005-03-21-10:42:39</Time>
    <Entity>{Domain lithium_ns:.testdomain}</Entity>
    <ResultStatus>
      <Code>1</Code>
      <Status>Response</Status>
      <Text>{Show Success} {Examination of attributes shows}</Text>
    </ResultStatus>
    <Director>lithium_ns:.temp.lithium_director</Director>
    <Arguments>
      <Attributes>
        <Owner__ID tempName="Owner ID">
          <Value>rta</Value>
          <ReasonCode>Available</ReasonCode>
        </Owner__ID>
        <Directory tempName="Directory">
          <Value>/home/rta</Value>
          <ReasonCode>Available</ReasonCode>
        </Directory>
        <Domain__category tempName="Domain category">
          <Value></Value>
          <ReasonCode>NotAvailable</ReasonCode>
        </Domain__category>
      </Attributes>
    </Arguments>
  </OK>
  <OK>
    <Time>2005-03-21-10:42:39</Time>
    <Entity>{Domain lithium_ns:.testdomain1}</Entity>

```

```
<ResultStatus>
  <Code>1</Code>
  <Status>Response</Status>
  <Text>{Show Success} {Examination of attributes shows}</Text>
</ResultStatus>
<Director>litium_ns:.temip.litium_director</Director>
<Arguments>
  <Attributes>
    <Owner__ID tempName="Owner ID">
      <Value>hugo</Value>
      <ReasonCode>Available</ReasonCode>
    </Owner__ID>
    <Directory tempName="Directory">
      <Value>/home/hgo</Value>
      <ReasonCode>Available</ReasonCode>
    </Directory>
    <Domain__category tempName="Domain category">
      <Value></Value>
      <ReasonCode>NotAvailable</ReasonCode>
    </Domain__category>
  </Attributes>
</Arguments>
</OK>
<ResultCount>2</ResultCount>
<ErrorFlag>>false</ErrorFlag>
</TeMIPResults>
```



# Chapter 4: OVSA FM

This chapter explains the composition and roles of the OVSA FM classes and provides details of their attributes and directives.

## Overview

The OVSA FM is the central and only component of the TeMIP side of *TeMIP Liaison*. It provides the following functionality to its clients:

- To show and start workflows on a given Service Activator instance
- To show request queues currently existing on a given Service Activator instance
- To show jobs currently running on a given service activator instance
- To supply case-packet variables to a job waiting for interaction.
- To stop currently running jobs.

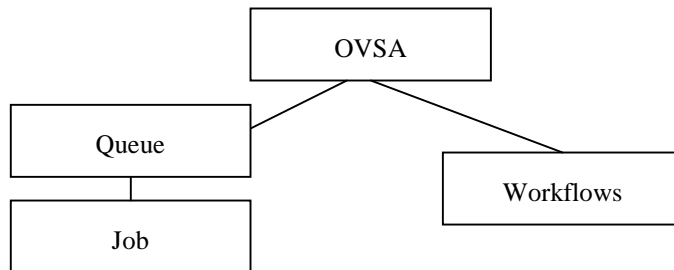
The OVSA FM consists of a self-management interface represented by class OVSA and a service interface containing four classes. These classes are described in the following sections.

## Service interface

Below is shown a figure of the Service interface. One global class exist, the OVSA class. An instance of this class represents an OVSA server. All child entities are proxies for objects on the OVSA server.

This hierarchy fully corresponds to the Service Activators logic. So, each OVSA instance has a number of workflows which can be started. The same workflow can be started unlimited number of times. The started workflow is named a “Job”. Each job belongs to a “Queue”.

**Figure 3, TeMIP model of OVSA FM Service Interface**



## OVSA

The entity represents a single Service Activator server. It contains the information internally used by the OVSA FM to connect to the Service Activator server. It is first when an instance of this class is created for a given Service Activator server it is possible to send request or to receive request from the Service Activator server.

The instances of the global class are persisted in a local MIR. Note this is the only class which is persistent on the Service interface.



## OVSA class attributes

**Table 1 OVSA Identifier Attributes**

Identifiers			
Attribute Name	Data type	Settable	Description
<b>id</b>	FullName	No	OVSA instance name

**Table 2 OVSA Characteristic Attributes**

Characteristics			
Attribute Name	Data type	Settable	Description
<b>description</b>	Latin1String	Yes	Description of the OVSA server instance. <i>Default is empty string.</i>
<b>hostname</b>	Latin1String	Yes	The name of the host of the OVSA server.
<b>portNumberMWFM</b>	Unsigned32	Yes	The MWFM listener port for the OVSA server <i>No default</i>
<b>portNumberRESMGR</b>	Unsigned32	Yes	The RESMGR listener port for the OVSA server <i>No default</i>
<b>portNumberWEB</b>	Unsigned32	Yes	The web listener port for the OVSA server <i>No default</i>
<b>userId</b>	Latin1String	Yes	The user id which is used when sending messages to the web/soap interface used in show directive calls <i>No default</i>
<b>password</b>	Latin1String	Yes	The password which is used when sending messages to the web/soap interface used in the show directive calls <i>No default</i>
<b>cidMWFM</b>	Latin1String	Yes	The connection id used for the connection to the MWFM <i>No default</i>
<b>cidRESMGR</b>	Latin1String	Yes	The connection id used for the connection to the RESMGR <i>No default</i>

## OVSA class directives

**Table 3 OVSA class Directives**

Directives		
<b>Show</b>	Standard responses / exceptions	
<b>Parameters:</b> None		
Examines the specified attribute or attributes partition and returns their values. Only non wild-carded calls are possible.		
<b>Create</b>	<b>Responses:</b>	<b>Exceptions:</b>
<b>Parameters:</b> description (Optional), hostname (Mandatory), portNumberMWFM (Mandatory), portNumberRESMGR (Mandatory), portNumberWEB (Mandatory), userId (Mandatory), password (Mandatory), cidMWFM (Mandatory), cidRESMGR (Mandatory)	Create Success	Argument Cid Error  Argument Cid Unique Error  Create Already Exists  Argument Missing
<p>Creates new OVSA instance. The arguments hostname, portNumberWEB, portNumberMWFM, portNumberRESMGR, cidMWFM and cidRESMGR must be unique for each OVSA instance.</p> <p>If hostname or one of cidMWFM or cidRESMGR is used by another entity the exception “Argument Cid Error” is returned.</p> <p>If cidMWFM or cidRESMGR are not unique then the exception “Argument Cid Unique Error” is returned.</p> <p>If the entity already exist then the exception “Create Already Exists” is returned.</p> <p>If one of the mandatory arguments are missing the exception “Argument Missing” is returned.</p>		
<b>Delete</b>	<b>Responses:</b>	<b>Exceptions:</b>
<b>Parameters:</b> None	Delete Success  Delete Nothing	Unable to Complete Operation  No Such Entity
<p>The OVSA FM checked for outstanding jobs to or from the deleted instance. If there are any outstanding jobs the OVSA instance can't be deleted and the common exception “Unable to Complete Operation” is returned. Otherwise the instance is deleted and a successful response is returned back to the caller.</p> <p>If the entity does not exists the common exception “No Such Entity” is returned.</p>		

## Queue

A queue entity represents a queue on the Service Activator server. The queue instances are always retrieved from the Service Activator server.

Two queues are always presented – “Running Jobs” and “Scheduled jobs”.

## Queue class attributes

**Table 4 Queue Identifier Attributes**

Identifiers			
Attribute Name	Data type	Settable	Description
name	Latin1Name	No	Job queue name

## Queue class directives

**Table 5 Queue class Directives**

Directives	
Show	Standard responses / exceptions
Parameters: None	
All queue instances are retrieved directly from the corresponding OVSA instance.	
Both wild-carded and non wild-carded calls are supported. Partially wild-carded calls are not supported.	

## Job

A job entity represents a job currently running on a Service Activator server. The Job instances are retrieved directly from the Service Activator server. The directives of this class provide the capability to show all the jobs currently running on the Service Activator server, to stop a job and to interact with a job, i.e. provide case-packet variables to a job waiting in a queue.

## Job class attributes

**Table 6 Job Identifier Attributes**

Identifiers			
Attribute Name	Data type	Settable	Description
jobId	SimpleName	No	The id of an active job

**Table 7 Job Characteristic Attributes**

<b>Characteristics</b>			
<b>Attribute Name</b>	<b>Data type</b>	<b>Settable</b>	<b>Description</b>
<b>workflow</b>	Latin1String	No	The name of the workflow this job is executing <i>No default</i>
<b>status</b>	Latin1String	No	Status of the job.  No default
<b>startTime</b>	BinAbsTime	No	The start time of the job.  <i>No default</i>
<b>postTime</b>	BinAbsTime	No	The time when the job was posted  <i>No default</i>
<b>scheduleTime</b>	BinAbsTime	No	The time when the job should be started. Only set if the job belongs to the queue “Scheduled Jobs”  <i>No default</i>
<b>endRepeating</b>	BinAbsTime	No	The time when the repeating job should end. Only set if the job belongs to the queue “Scheduled Jobs” and it is repeatable job  <i>No default</i>
<b>repeatingPeriod</b>	Latin1String	No	The repeating period of the job. Only set if the job belongs to the queue “Scheduled Jobs” and it is repeatable job  <i>No default</i>
<b>username</b>	Latin1String	No	User name of the user who posted the Scheduled Job.  <i>No default</i>
<b>groupId</b>	Latin1String	No	A string which can be used to group scheduled jobs together.  <i>No default</i>
<b>step</b>	Latin1String	No	Which node the job is currently running.  <i>No default</i>
<b>description</b>	Latin1String	No	The workflow description.

## Job class directives

**Table 8 Job class Directives**

Directives		
<b>Show</b>	Standard responses / exceptions	
<b>Parameters:</b> None		
<p>Examines the specified attribute or attributes partition and returns their values.</p> <p>The directive returns the runtime information about jobs currently running on a Service Activator server.</p> <p>Both wild-carded and non wild-carded calls are possible. Partial wild-carding is not supported.</p>		
<b>Stop</b>	<b>Responses:</b>	<b>Exceptions:</b>
<b>Parameters:</b> userId (Mandatory), password(Mandatory)	Stop Job Success	Exception Argument Missing
<p>Kills running job.</p> <p>Two mandatory arguments must be provided – userId and password – indicates as which user this job will be killed.</p> <p>If one of the arguments is not present then the exception “Argument Missing” is returned.</p> <p>If the Service Activator server returns an exception then exception “Exception” is returned by the call.</p>		
<b>Load</b>	<b>Responses:</b>	<b>Exceptions:</b>
<b>Parameters:</b> casePacket (Mandatory), userId (Mandatory), password (Mandatory)	Load Job Success	Exception Argument Missing Field Missing
<p>Sends case-packet variables to the job, i.e. it is the same as if an operator interact with the job on OVSA’s GUI.</p> <p>Three mandatory arguments must be provided to the call:</p> <ul style="list-style-type: none"> <li>• casePacket – contains the name-value pairs of the case-packet variables. The argument is of type SequenceOfRecords where each record has two fields – name and value. First one, is the name of the case-packet variable, the second, is its value. Both fields are of type Latin1 String.</li> <li>• userId and password – indicates for which user this interaction is done as.</li> </ul> <p>If one of the arguments is missing then the exception “Argument Missing” is returned.</p> <p>If an error occurred on the Service Activator server when interacting with the job then the exception “Exception” is returned by the call.</p> <p>If either name or value is missed at least in one of the name-value pairs then the exception “Field Missing” is returned by the call.</p>		

## Workflow

The workflow entity represents a workflow on the Service Activator server. The show directive shows one or all the workflows on the Service Activator server and the start directive starts a workflow.

### Workflow class attributes

**Table 9 Workflow Identifier Attributes**

Identifiers			
Attribute Name	Data type	Settable	Description
<b>workflow</b>	SimpleName	No	The name of the workflow

**Table 10 Workflow Characteristic Attributes**

Characteristic			
Attribute Name	Data type	Settable	Description
<b>description</b>	Latin1String	No	Workflow description

### Workflow class directives

**Table 11 Workflow class Directives**

Directives		
<b>Show</b>	Standard responses / exceptions	
<b>Parameters:</b> None		
Shows the workflow(s) which exists on the Service Activator server. Both non wild-carded and wild-carded calls are supported. Partial wild-carding is not supported.		
<b>Start</b>	<b>Responses:</b>	<b>Exceptions:</b>
<b>Parameters:</b> casePacket (Optional), userId (Mandatory), password (Mandatory)	Start Job Success	Exception Argument Missing
Starts a workflow on the Service Activator server.		
Three arguments exist for the directive:		
<ul style="list-style-type: none"> <li>• casePacket – is used to pass start-up data to the workflow. The argument is of type Latin1String. The data will be saved in a file on the Service Activator server and the workflow will be started with the case packet variable message_file.</li> <li>• userId and password – indicates for which user the workflow should be started. Both arguments are of type Latin1String.</li> </ul>		

If one of the mandatory arguments is missing then the exception “Argument Missing” is returned.

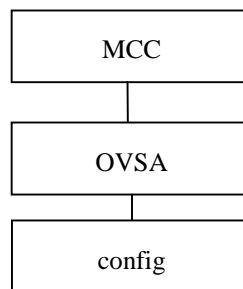
If an error occurred on the Service Activator server during the workflow start-up then the exception “Exception” is returned by the call.

## Self-management interface

The OVSA FM supports a number of configuration parameters, settable as characteristics attributes on its TeMIP self-management interface, i.e. on the “mcc 0 ovsa config” entity.

The self-management interface has the following hierarchy:

**Figure 4, OVSA FM Self Management classes**



## OVSA

### OVSA directives

All TeMIP Framework management modules must support certain required directives (Show and Test). Class OVSA supports only these directives.

**Table 12 OVSA class Directives**

Directives	
<b>Show</b>	Standard responses / exceptions
<b>Parameters:</b> None	
Examines the specified attribute or attributes partition and returns their values.	
<b>Test</b>	Standard responses / exceptions
<b>Parameters:</b> None	
Verifies that the component is correctly enrolled into the TeMIP Framework system dispatch table.	

## OVSA config class

The instance holds the common configuration of the OVSA FM. This configuration is used to configure communication links to all Service Activator servers.

When a configuration parameter is set it does not take effect until OVSA FM is restarted.

### OVSA config class attributes

The values of the attributes *portNumber*, *connConfirmTimeout* and *confirmTimeout* are persisted in the local MIR.

**Table 13 OVSA config Characteristic Attributes**

Characteristics			
Attribute Name	Data type	Settable	Description
<b>Component Identification</b>	Latin1String	No	Describe the Identification of the OVSA FM
<b>Component Version</b>	Version	No	Describe the version number of the OVSA FM
<b>portNumber</b>	Unsigned32	Yes	The listener port number, i.e. this is the port that all Service Activator servers will connect to when communication with TeMIP.  <i>Default is 4050</i>
<b>connConfirmTimeout</b>	Unsigned32	Yes	This is the number of milliseconds that the FM will wait until again try to send the connect request message. This parameter is used when requests are sent from Service Activator server to TeMIP. Normally this value should never be changed. However in rare cases where a bad network connection exists it would help to increase this value.  <i>Default is 10000</i>
<b>confirmTimeout</b>	Unsigned32	Yes	This is the number of milliseconds that the FM will wait until again try to send the request message. The same rule apply to this attribute as for the attribute conConfirmTimeout.  <i>Default is 10000</i>

### OVSA config class directives

The class supports only two directives – show and set.



**Table 14 OVSA config class Directives**

Directives	
<b>Show</b>	Standard responses / exceptions
<b>Parameters:</b> None	
Examines the specified attribute or attributes partition and returns their values.	
<b>Set</b>	Standard responses / exceptions
<b>Parameters:</b> Attribute List	
Sets the configuration attributes. The attributes <code>portNumber</code> , <code>connConfirmTimeout</code> , and <code>confirmTimeout</code> are the only ones which can set. However the values will first be used next time the OVSA FM is started.	

# Chapter 5: TeMIP nodes

This chapter describes the best approaches to usage of TeMIP nodes as well as provides the examples of configuration of the nodes and processing of their results.

## Overview

Five nodes exist to provide the capability to execute directives and TCL expressions on a TeMIP director. Below is a brief description of each of them:

- TeMIPStartDirective - starts a directive, i.e. executes the directive call but do not wait for the call to complete. The call results are returned to the workflow one at a time as fast as they are returned from TeMIP. The results can be retrieved by the TeMIPNextResult node.
- TeMIPNextResult – returns the next result from a previously initiated start directive.
- TeMIPCancelDirective - cancels a started directive
- TeMIPExecuteDirective - executes a directive call, waits for the execution of the call to complete, and returns all resulting data.
- TeMIPExecTCL - executes a sequence of TCL commands on a TeMIP director and returns the result of the last command to the workflow.

Detailed information about each node can be found in the *Workflows and the Micro-Workflow Manager* document.

The TeMIP nodes are intended for use in workflows to execute directives or TCL commands on a TeMIP director where the execution of the directive or commands is limited in time. The main reason is that the workflow threads - a critical resource - are NOT released during the node execution.

If it takes long time to execute the directive or TCL commands, or if a transaction is necessary, then use the TeMIPLiaison plug-in, which is described in Chapter 6 of this document.

A good practice is to only query for information in nodes and make modification, creating, and deletion through the TeMIPLiaison plug-in.

Which node to use when depends on what to execute and what is the purpose with the data returned.

Use the TeMIPExecuteDirective in the following situations:

- For directive execution which always return a single response
- For directive execution which can return multiple responses where all of them are required in the workflow before any further workflow action can be taken.

Use the TeMIPStartDirective and TeMIPNextResult in the following situations:

- For directive execution where a specific action will be taken for each call result returned
- For directive execution where the workflow searches for a specific instance is needed and this is not the identifier. When the instance is found the rest of the result data is not needed. In this situation the TeMIPCancelDirective should be called when the instance is found to cancel directive for further execution.

Use the TeMIPExecTCL in the following situations:

- When execution of TCL commands are wanted
- When any of the other TTS features are wanted

---

## NOTE

It is always possible to use the nodes TeMIPStartDirective - TeMIPNextResult instead of only using the TeMIPExecuteDirective. However this is not good practice as this will unnecessarily complicate the workflow and the workflow execution time will be slower.

---

The following sections will give some examples of how to use the different nodes.

## TeMIPExecuteDirective Examples

The RET\_VALUE is also set for the TeMIPExecuteDirective node; however for this node the RET\_VALUE can have four values. The node will set the value to:

- |   |   |
|---|---|
| 0 | if the directive returns a response                                     |
| 1 | if an error occur on one of the sides during execution of the directive |
| 2 | if the directive returns a SpecializedException                         |
| 3 | if the directive returns a CommonException.                             |

The RET\_VALUE is then easy to use in you workflow logic to verify if directive has been executed successfully or not and in case of exception what kind of an exception is received. This could be the case where a workflow wants to verify if a TeMIP instance exists or not. In this case would a show directive call done as a non wild-carded show directive call on the identifier partition for the given instance return a RET\_VALUE of 0 if the entity exists (a TeMIP response is returned) and a RET\_VALUE of 3 if the entity did not exists (CommonException “No Such Entity”).

If further information is needed in the workflow from the directive result the case-packet variables provided to the node for the parameters *xml\_result* and *xml\_results* must be analyzed. The easiest way to do this is by using the XMLMapper node (see description in [MWFM]).

The case-packet variable for the *xml\_result* parameter will always contain the first response returned from the directive and the XML syntax will follow the syntax defined for the element *TeMIPResult*. The case-packet variable for the *xml\_results* parameter will always contain all responses returned from the directive and the XML syntax will follow the syntax defined for the element *TeMIPResults*. See chapter 3 for further information.

## Non wild-carded Show Call

This example makes a “show” directive for the “characteristic” partition on the entity `mcc 0 application ovsa`.

To execute this directive the node can be configured the following way:

Figure 5, TeMIPExecuteDirective node parameters

Name	Value
connection_id	MWFM_sa0_director0
directive_exp	show {{mcc 0}} {application ovsa}} char
user_name	test
xml_result	variable:xmlResultCPV
xml_results	variable:xmlResultsCPV

Note that as far as the directive has no arguments there is no parameter “arguments” in the node configuration.

The parameter *directive\_exp* is used in this example to set the verb, entity and partition of the directive. The call parameters *verb*, *entity* and *partition* are omitted.

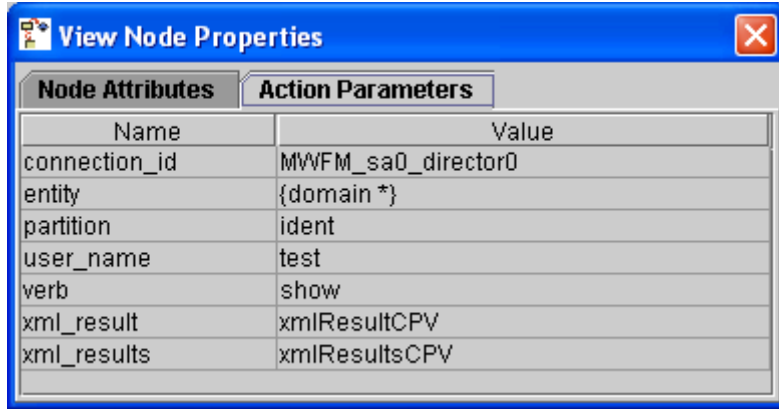
The node will set case-packet *xmlResultCPV* to the following string:

```
<OK>
  <Time>2005-03-24-12:54:43</Time>
  <Entity>{MCC lithium_ns:.temip.litium_director} {APPLICATION ovsa}</Entity>
  <ResultStatus>
    <Code>1</Code>
    <Status>Response</Status>
    <Text>{Show Success} {Examination of Attributes Shows:}</Text>
  </ResultStatus>
  <Director>lithium_ns:.temip.litium_director</Director>
  <Arguments>
    <Attributes>
      <Application__Type temipName="Application Type">
        <Value>MM</Value>
        <ReasonCode>Available</ReasonCode>
      </Application__Type>
      <Module__Type temipName="Module Type">
        <Value>FM</Value>
        <ReasonCode>Available</ReasonCode>
      </Module__Type>
      <Filename temipName="Filename">
        <Value>/usr/opt/temip/bin/tts_mm</Value>
        <ReasonCode>Available</ReasonCode>
      </Filename>
      <Arguments temipName="Arguments">
        <Value></Value>
        <ReasonCode>Available</ReasonCode>
      </Arguments>
      <Automatic__Startup temipName="Automatic Startup">
        <Value>True</Value>
        <ReasonCode>Available</ReasonCode>
      </Automatic__Startup>
      <Trace__File temipName="Trace File">
        <Value>/var/opt/temip/trace/ovsa.log</Value>
        <ReasonCode>Available</ReasonCode>
      </Trace__File>
      ...
      ...
    </Attributes>
  </Arguments>
</OK>
```

## Wild-carded Show Call

In following example the TeMIPExecuteDirective node is configured to execute wild-carded *show* of *domains* on partition *identifiers* (see Figure 1).

Figure 6, TeMIPExecuteDirective node parameters



Name	Value
connection_id	MWFM_sa0_director0
entity	{domain *}
partition	ident
user_name	test
verb	show
xml_result	xmlResultCPV
xml_results	xmlResultsCPV

The node places all the responses into the case-packet variable *xmlResultsCPV*. The first response is placed into the case-packet *xmlResultCPV*.

After successful execution of the node the *xmlResultsCPV* will contain a below string:

```
<TeMIPResults>
  <Handle>1111397617433#TeMIPStartDirective#-552608615852567522</Handle>
  <ResponseNumber>1</ResponseNumber>
  <OK>
    <Time>2005-03-24-13:13:50</Time>
    <Entity>{Domain lithium_ns:.jane}</Entity>
    <ResultStatus>
      <Code>1</Code>
      <Status>Response</Status>
      <Text>{Show Success} {Examination of attributes shows}</Text>
    </ResultStatus>
    <Director>lithium_ns:.temp.litium_director</Director>
    <Arguments>
      <Attributes>
        <DomainName tempName="DomainName">
          <Value>lithium_ns:.jane</Value>
          <ReasonCode>Available</ReasonCode>
        </DomainName>
      </Attributes>
    </Arguments>
  </OK>
  <OK>
    <Time>2005-03-24-13:13:50</Time>
    <Entity>{Domain lithium_ns:.janr}</Entity>
    <ResultStatus>
      <Code>1</Code>
      <Status>Response</Status>
      <Text>{Show Success} {Examination of attributes shows}</Text>
    </ResultStatus>
    <Director>lithium_ns:.temp.litium_director</Director>
    <Arguments>
      <Attributes>
        <DomainName tempName="DomainName">
          <Value>lithium_ns:.janr</Value>
          <ReasonCode>Available</ReasonCode>
        </DomainName>
      </Attributes>
    </Arguments>
  </OK>
</TeMIPResults>
```

```

        </Attributes>
    </Arguments>
</OK>
.....
.....
    <ResultCount>8</ResultCount>
    <ErrorFlag>>false</ErrorFlag>
</TeMIPResults>

```

There are eight domains (<ResultCount>8</ResultCount>) on the TeMIP server. So, eight responses for this directive call are returned the Domain FM. Correspondently TeMIPResults message contains eight “OK” elements – one per response.

The *xmlResultCPV* will be set to the string containing the first “OK” element:

```

<OK>
  <Time>2005-03-24-13:13:50</Time>
  <Entity>{Domain lithium_ns:.jane}</Entity>
  <ResultStatus>
    <Code>1</Code>
    <Status>Response</Status>
    <Text>{Show Success} {Examination of attributes shows}</Text>
  </ResultStatus>
  <Director>lithium_ns:.temp.lithium_director</Director>
  <Arguments>
    <Attributes>
      <DomainName tempName="DomainName">
        <Value>lithium_ns:.jane</Value>
        <ReasonCode>Available</ReasonCode>
      </DomainName>
    </Attributes>
  </Arguments>
</OK>

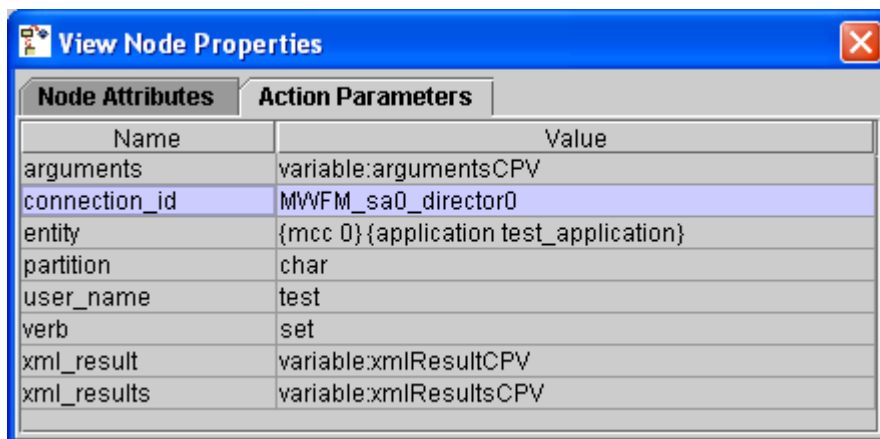
```

## Action and modification calls

In the following example we will configure the TeMIPEXecuteDirective node to “set” the “characteristic” attribute “Trace File” of the application “test\_application” ({mcc 0} {application test\_application}) to value “/tmp/test\_application.log”. The directive call returns a single response or an exception. This is why the response message will contain only one “OK” element and the value of element “ResultCount” will be equal to “1”.

The node will, by using the below node parameters, place the element “OK” of the response message - in the case-packet variable *xmlResultCPV* and the entire response message in the case-packet variable *xmlResultsCPV*.

Figure 7, TeMIPEXecuteDirective node parameters



Name	Value
arguments	variable:argumentsCPV
connection_id	MWFM_sa0_director0
entity	{mcc 0}{application test_application}
partition	char
user_name	test
verb	set
xml_result	variable:xmlResultCPV
xml_results	variable:xmlResultsCPV

The case-packet variable *argumentsCPV*, provided in the node parameter *argument*, must contain the following XML string:

```
<Arguments>
  <Attributes>
    <Trace__File tempName="Trace File">
      <Value>
        /tmp/test_application.log
      </Value>
    </Trace__File>
  </Attributes>
</Arguments>
```

Or to the equivalent string:

```
<Arguments>
  <Attribute__Values tempName="Attribute Values">
    <Value>
      <AttribList>
        <Trace__File tempName="Trace File">
          <Value>
            /tmp/test_application.log
          </Value>
        </Trace__File>
      </AttribList>
    </Value>
  </ Attribute__Values>
</Arguments>
```

In the example the case-packet variable *argumentsCPV* is used to pass this string to the node, although it is possible to set the node parameter "arguments" to a constant value containing the same string.

---

## NOTE

All of the parameters of the node *TeMIPEXecuteDirective* can be set both to constant values and variables. For more information see the node description in the document [MWFM].

---

As a result of execution of the node, the *RET\_VALUE* will be set to "0" and the *xmlResultCPV* will be set to the XML string:

```
<OK>
  <Time> Thu Oct 7 09:48:46</Time>
  <Entity>
    {MCC lithium_ns:.temp.litium_director} {APPLICATION ovsa}
  </Entity>
  <ResultStatus>
    <Code>1</Code>
    <Status>Response</Status>
    <Text>
      {Set Success} {Modification(s) completed successfully.}
    </Text>
  </ResultStatus>
  <Director> lithium_ns:.temp.litium_director</Director>
  <Arguments>
    <Attributes>
      <Trace__File tempName="Trace File">
        <Value>/tmp/test_application.log </Value>
        <ReasonCode>Available</ReasonCode>
      </Trace__File>
    </Attributes>
```

```
</Arguments>
</OK>
```

And the *xmlResultsCPV* will be set to the string:

```
<TeMIPResults>
  <Handle>1111397617433#TeMIPStartDirective#-552608615852567522</Handle>
  <ResponseNumber>1</ResponseNumber>
  <OK>
    <Time> Thu Oct 7 09:48:46</Time>
    <Entity>
      {MCC lithium_ns:.temip.litium_director} {APPLICATION ovsa}
    </Entity>
    <ResultStatus>
      <Code>1</Code>
      <Status>Response</Status>
      <Text>
        {Set Success} {Modification(s) completed successfully.}
      </Text>
    </ResultStatus>
    <Director> lithium_ns:.temip.litium_director</Director>
    <Arguments>
      <Attributes>
        <Trace__File temipName="Trace File">
          <Value>/tmp/test_application.log </Value>
          <ReasonCode>Available</ReasonCode>
        </Trace__File>
      </Attributes>
    </Arguments>
  </OK>
  <ResultCount>1</ResultCount>
  <ErrorFlag>>false</ErrorFlag>
</TeMIPResults>
```

The case-packet variable *xmlResultsCPV* contains the complete response also in the case where only one response is returned and there will be only one “OK” element in the message and the “ResultsCount” is “1”. An exception was not returned from the directive call, so the element “ErrorFlag” has the value “false”.

In the case where directive call has returned a *CommonException* or *SpecializedException* the *xmlResultsCPV* would have contained this exception and the element “ErrorFlag” would have had the value “true”. Below is an example for a *CommonException*:

```
<TeMIPResults>
  <Handle>1111397617433#TeMIPStartDirective#-552608615852567522</Handle>
  <ResponseNumber>1</ResponseNumber>
  <OK>
    <Time>2005-03-21-13:52:54</Time>
    <Entity>{Domain lithium_ns:.notexist}</Entity>
    <ResultStatus>
      <Code>3</Code>
      <Status>CommonException</Status>
      <Text>{No Such Entity} {No such entity: Domain
lithium_ns:.notexist}</Text>
    </ResultStatus>
    <Director>lithium_ns:.temip.litium_director</Director>
  </OK>
  <ResultCount>1</ResultCount>
  <ErrorFlag>>true</ErrorFlag>
</TeMIPResults>
```

The *xmlResultCPV* will be set to the “OK” part of the string.



---

## NOTE

The value of the element <Entity> is the same as the one set for the node parameter “entity”, except that TNS namespace now has been added. This is always the case when the entity is returned from TeMIP.

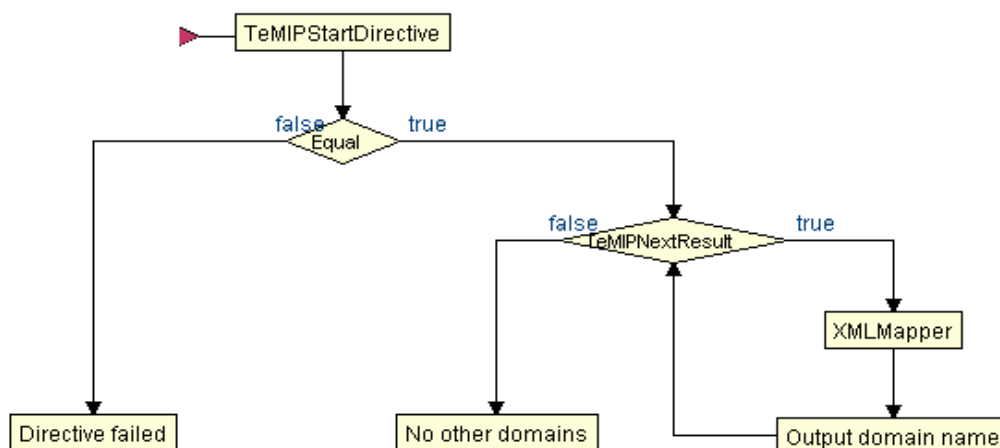
---

## TeMIPStartDirective and TeMIPNextResult Example

Let’s assume that in your workflow you need to execute an action for each of the domain instances existing on a TeMIP director. This is the case when the combination of the nodes TeMIPStartDirective and TeMIPNextResult should be applied. In the example workflow, shown in the diagram below, the name of the domain is just written as a message to avoid complexity. However, in a real workflow it could very likely be that another directive call should be made with some data from the first directive call. In this case would the node “Output domain name” be substituted with the new directive call.

The Example workflow starts the directive (TeMIPStartDirective node) and then verifies the RET\_VALUE (node Equal). If RET\_VALUE equals to “1” (i.e. an error occurred during the directive start-up) then the workflow places the message “Directive failed” into the queue “Domains” (node “Directive failed”) and finishes, otherwise, if the directive has been successfully started then the workflow loops through all of the responses using the node TeMIPNextResult, extracts the DomainName attribute of each of the domains by means of the XMLMapper node and put a message to the queue “Domains” (node “Output domain name”). If the last response has already been retrieved and displayed then the call to the TeMIPNextResult node will result in that the workflow follows the false branch. The workflow will also follow the false branch in the case where the directive returns a Common or Specialized exception. In this case, the workflow put a message - “No other domains” - to the queue “Domains”.

**Figure 8, Executing wild-carded show with TeMIPStartDirective and TeMIPNextResult nodes.**



The following two figures show the configuration of the nodes of this workflow.

Figure 9, TeMIPStartDirective node parameters

Name	Value
connection_id	MWFM_sa0_director0
entity	{domain *}
handle	variable:handleCPV
partition	ident
user_name	test
verb	show

Figure 10, TeMIPNextResult node parameters

Name	Value
handle	handleCPV
xml_result	xmlResultCPV

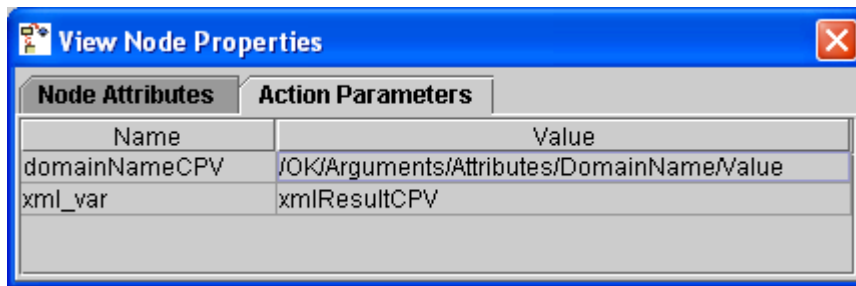
Parameter “handle” in both TeMIPStartDirective (see figure 1) and TeMIPNextResult (see figure 2) points to the same case-packet variable – *handleCPV*. TeMIPStartDirective will place the directive call identifier into the *handleCPV*, on the other hand, the TeMIPNextResult will use it to retrieve the responses of exactly the directive started by TeMIPStartDirective node.

The string placed into *xmlResultCPV* by TeMIPNextResult on each of iterations will have a next structure:

```
<OK>
  <Arguments>
    <Attributes>
      <DomainName tempName="DomainName">
        <Value>
          A domain name including TNS name space
        </Value>
      </DomainName>
    </Attributes>
  </Arguments>
</OK>
```

To extract the value of the attribute DomainName from the response placed in the case-packet variable *xmlResultCPV* the node XMLMapper is configured as described in the figure below.

Figure 11, XMLMapper node parameters



The node extracts the value of the attribute DomainName from the element “/OK/Arguments/Attributes/DomainName/Value” and places it in the case-packet variable *domainNameCPV*. The value of the case-packet variable *domainNameCPV* is then put in a message by the node “Output domain name”.

Because of the DomainName is a simple TeMIP type - FullName - the value is encoded into XML directly – without additional wrapping.

## TCL execution

If TeMIPStartDirective, TeMIPNextResult and TeMIPExecuteDirective nodes do not completely satisfy your particular needs or if you want to move some part of the calculations from your workflow to TeMIP server you can then use the node TeMIPExecTCL. Place an arbitrary sequence of TCL expressions into the node parameter *tcl\_exp* or create a file with the sequence of expressions and indicate the file name with full path in parameter *tcl\_file*. The node will execute the sequence of TCL commands and place the result of last command into the case-packet variable indicated for the parameter *result*.

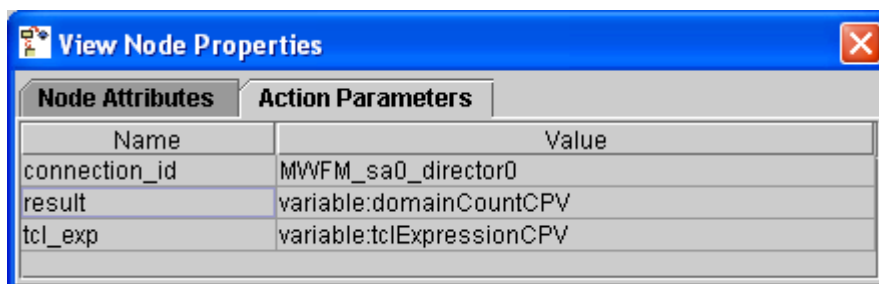
---

### NOTE

The sequence of TCL expressions can contain both standard TCL commands and TTS commands (see document [TTS] for more information).

---

Figure 12, TeMIPExecTCL node parameters



The TCL commands are passed to the node in the case-packet variable *tclExpressionCPV*. The result of the execution of the last command is placed into the case-packet variable *domainCountCPV*.

The TCL commands given in the parameter *tclExpressionCPV* looks the following way:

```

set dNumber 0

#construct the call
set call [temp::call setup show {{domain *}} ident]

set more 1
#loop through domains
while {$more != 0 } {
  set res [$call call]
  if {$res == "Response"} {
    set dNumber [expr {$dNumber + 1}]
    set more [$call more]
  } else {
    break;
  }
}
#return the number of domains
set dNumber

```

The value of variable *dNumber* will be returned to the workflow and placed into case-packet *domainCountCPV*.

## Variable substitution

The majority of the parameters in the TeMIP nodes support substitution of case-packet variables. Any occurrence of *%variable%* within the parameter value is replaced with the value of the corresponding case-packet variable. If the corresponding case-packet variable is not defined in the workflow, then the occurrence of *%variable%* is left in the parameter value without changes.

For example, you might want to define the partition of the directive call at the runtime. In this case, the parameter *directive\_exp* can be constructed like: *show {{mcc 0}} %partitionCPV%*. The occurrence of *%partitionCPV%* is replaced with the value of the *partitionCPV* case-packet variable.

Variables are also substituted in the content of the directive or TCL file (the parameter *directive\_file* in the nodes TeMIPStartDirective and TeMIPEXecuteDirective and parameter *tcl\_file* defined in the node TeMIPEXecTCL).

## Template files

The definition of the directive or the sequence of TCL commands to execute can be placed into a file – template file. The default location for these files is \$ACTIVATOR\_ETC/template\_files.

This feature is extremely useful when a complicated directive call is constructed with lot of attributes or arguments or in the case of a large sequence of TCL commands. This feature can be combined with variable substitution and make it even more useful.

To execute the directive using the node TeMIPEXecuteDirective the file name with the message “ExecuteDirective” should be given in the parameter *directive\_file*. To start the directive using TeMIPStartDirective node the message “StartDirective” should be placed into template file

The sequence of TCL commands is placed into the file without wrapping.

For example, the content of the directive file to execute a wild-carded “show” of domain “characteristics” must be:

```

<ExecuteDirective>
  <User>test</User>
  <Expression>
    <Verb>show</Verb>
    <Entity>{domain *}</Entity>
    <Partition>char</Partition>
  </Expression>
</ExecuteDirective>

```

To set the attribute “Trace File” of the domain “liaisontest” to the value “/tmp/liaisontest.log” the content of the fill must be:

```
<ExecuteDirective>
  <User>test</User>
  <Expression>
    <Verb>set</Verb>
    <Entity>{domain liaisontest}</Entity>
    <Partition>char</Partition>
    <Arguments>
      <Attributes>
        <Trace__File tempName = "Trace File">
          <Value>
            /tmp/liaisontest.log
          </Value>
        </Trace__File>
      </Attributes>
    </Arguments>
  </Expression>
</ExecuteDirective>
```

As mentioned in the previous section the substitution of case-packet variables is also done in the content of the template file. So, for example, if you want to do the modification of the domain attributes more dynamic, you can replace the domain instance name as well as modified attribute name and partition in previous example with the names of the correspondent case-packets as follows:

```
<ExecuteDirective>
  <User>test</User>
  <Expression>
    <Verb>set</Verb>
    <Entity>{domain %domainName%}</Entity>
    <Partition>%attributePartition%</Partition>
    <Arguments>
      <Attributes>
        <%attributeTag% tempName = "%attributeName%">
          <Value>
            %attributeValue%
          </Value>
        </%attributeTag%>
      </Attributes>
    </Arguments>
  </Expression>
</ExecuteDirective>
```

# Chapter 6: TeMIPLiaison plug-in

This chapter describes the usage of the TeMIPLiaison plug-in. The description of the plug-in can be found in the document *Developing Plug-Ins and Compound Tasks* and the *Javadoc* associated with the code.

## Overview

There are two ways to execute requests to a TeMIP director. The first one is by means of TeMIP workflow nodes; this is described in the previous chapter. As noted above, this approach is applicable for “easy” requests, which are processed fast without blocking the workflow for a long time. However, it does not guarantee the consistency of the system in the case of a failure.

The second one is the activations through the TeMIPLiaison plug-in. It should be used for long-lasting requests and in the case a transaction is required. First of all, the workflow thread is released during the activation which helps to use the system resources efficiently and secondly, the plug-in rollback mechanism ensures that the system is left in a consistent state in the case of a failure.

---

### NOTE

The workflow thread is released when the activation is performed by means of the Multi-Threaded Activation Module. In the case where the Simple Activation Module is used the workflow threads are blocked when activation is performed.

---

The plug-in contains two atomic tasks:

- **temip\_EXP\_directive** executes a TeMIP directive on a given TeMIP director and upload the result data to the workflow.
- **temip\_EXP\_tcl** evaluates a sequence of TCL commands on a given TeMIP director and upload the result data to the workflow.

Both atomic tasks support the same number of parameters. A description of the parameters and what data is uploaded to the workflow can be found in the document *Developing Plug-Ins and Compound Tasks* and the *Javadoc* associated with the code.

No locking is needed when sending requests to a TeMIP director, but an atomic task must have at least one parameter to lock on. The parameter JobId (the ID of the current job) has been added to both atomic tasks for this purpose. This does not cause any problems as only one activation can be done at a time in a workflow.

## Transaction rollback

Each atomic task is reversible. If the Resource Manager determines that an activation transaction needs to be rolled back, then any atomic tasks in a transaction that have been completed will be invoked again and told to undo their changes.

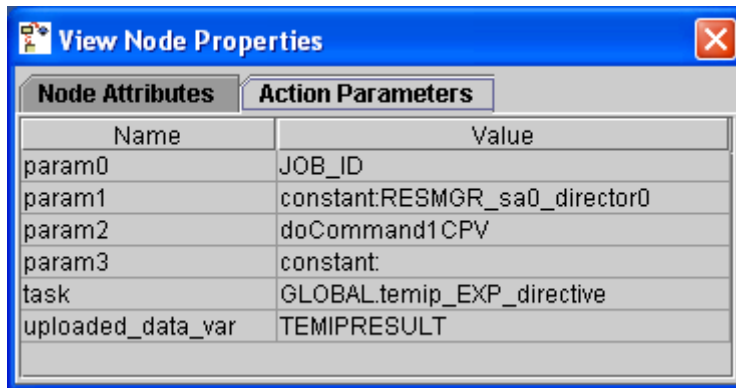
Accordingly, both atomic tasks of the TeMIPLiaison plug-in have two parameters the *do* and *undo* parameter. The first one indicates the command, which should be executed by the task and the second one the command which will be executed if the transaction is rolled back.

So, for example, if the task creates a domain in the “do” action, then it would be most probably to delete domain in the “undo” command.

## Temp\_EXP\_directive Example

Let's assume that your workflow should update a domain.

Figure 13, Parameters of the activation node to update the domain



The screenshot shows a window titled "View Node Properties" with two tabs: "Node Attributes" and "Action Parameters". The "Action Parameters" tab is active, displaying a table with two columns: "Name" and "Value".

Name	Value
param0	JOB_ID
param1	constant:RESMGR_sa0_director0
param2	doCommand1CPV
param3	constant:
task	GLOBAL.temip_EXP_directive
uploaded_data_var	TEMIPRESULT

Parameter *doCommand1CPV* contains the request message to set the “reference” attribute “Phone Number” of “testDomain1” to the value “123456789”:

```
<ExecuteDirective>
  <Expression>
    <Verb>set </Verb>
    <Entity>{domain testDomain1}</Entity>
    <Partition>ref</Partition>
    <Arguments>
      <Attributes>
        <Phone__Number temipName="Phone Number">
          <Value>123456789</Value>
        </Phone__Number>
      </Attributes>
    </Arguments>
  </Expression>
</ExecuteDirective>
```

Note that the “undo” command in this example is empty for the task “update domain”. This can be OK in a set directive as the one in this example, but in many other cases would this left the system in an inconsistent state.

The *uploaded\_data\_var* is set with the value TEMIPRESULT. The case-packet variable TEMIPRESULT will then after the activation contains the *TeMIPResults* string.

## Variable substitution

The variable substitution is done in the “do” and “undo” messages the same way as for the TeMIP nodes. The values of the case-packet variables which should be substituted into the messages are passed to both atomic tasks in the parameter “variables”.

In the example above would it be possible to replace the domain instance name and phone number with the names of case-packet variable as shown below:

```
<ExecuteDirective>
  <Expression>
    <Verb>set </Verb>
    <Entity>{domain %domainName%}</Entity>
    <Partition>ref</Partition>
    <Arguments>
      <Attributes>
        <Phone__Number temipName="Phone Number">
```

```
        <Value>%phoneNumber%</Value>
    </Phone__Number>
</Attributes>
</Arguments>
</Expression>
</ExecuteDirective>
```

The string passed to the task parameter “variables” should look this way:

```
<Variables>
  <domainName>testDomain1/domainName>
  <phoneNumber>123456789</phoneNumber>
</Variables>
```

If the *do* and the *undo* commands are provided in files then substitution of case-packet variables is done in the file content as well.

## Template files

The *do* and the *undo* command can also be provided by giving the name of a file which should be used as the parameter. The syntax when using this option is “**file:** xml\_filename”.

If a filename is specified, it must include the full pathname to the file.