ERICSSON

Simple Mediation

System Administration Guide

Proprietary Nature of Document

Each Ericsson document is prepared for the sole and exclusive use of the party or organization to which it is addressed. Therefore, Ericsson documents are proprietary and may not be made available to any other than the addressee or persons within the addressee's organizations who are designated to evaluate or implement the proposed application. Ericsson documents may be made available to other persons or organizations only with the permission of the Ericsson office issuing the documents.

Copyright

Document Revision History

Revision	Date	Updated By	Description
PA1		Ata ul Malik	Initial Draft
PA2		Ata ul Malik	Updated with SIM version, 3.3.1, 3.3.2, 4.1, 4.2, 5.1 Added 3.3.3
PA3		Ata ul Malik	Updated 3.2 Added 3.3
PA4		Ata ul Malik	Added 5.2. 5.3

Contents

1 Introduction

This document is intended to describe the Simple Mediation v3 (SIMv3) application, how it shall be installed, configured, and maintained. It will as well cover the supported functions and application design specifics and how to use it in operations.

The SIM application is used to collect date from one or many destinations. The application provides facility to filter, collect, and maintain backlog of data from remote machine (node) to local machine.

The protocols supported by SIM to data collection are:

1. SFTP

1.1 Readership

The intended audience for this document includes:

- Ericsson CU
- Ericsson Design, Installation, Integration, and testing personnel

2 Application Overview

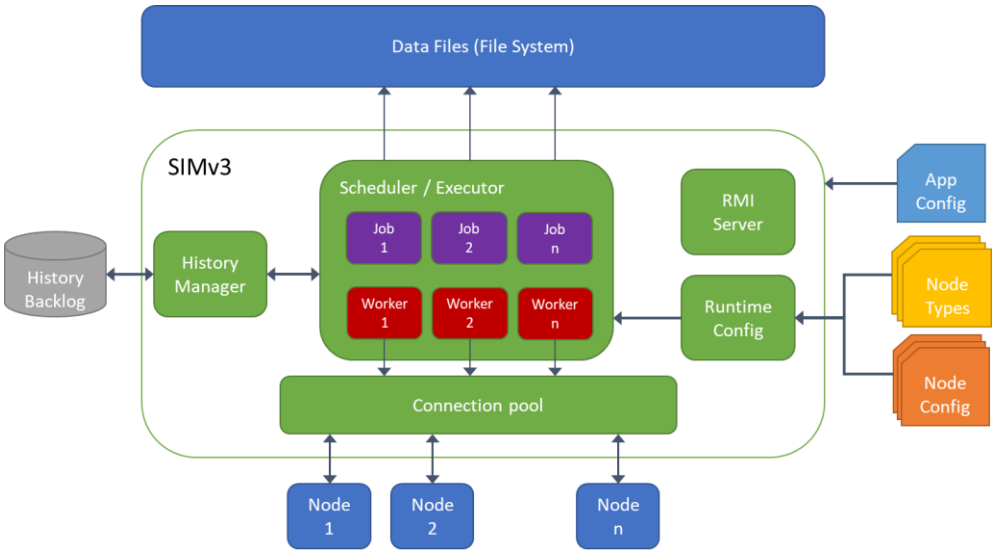
2.1 Overview

SIM is a lightweight, high performing, and easy to configure application that supports data collection for a defined group of nodes. Data is collected using SFTP protocol though SSH.

SIM supports following functionality:

1. Node types that define default values for jobs scheduling, data identification and transfer, and housekeeping of files on nodes
2. Runtime configuration for one or many nodes from which data will be collected
3. Scheduling facility based on configuration
4. Parallel collection of multiple files from multiple nodes to improve performance
5. Pre-checks before collection to make sure that system is ready to handle collection i.e., % free disk space
6. Housekeeping of files on remote system
7. Backlog management with ability to recover files from node (the file should exist on node)
8. A CLI to look at runtime statistics as well as add new node types

Below diagram shows high level architecture for SIM:



2.2 Application Logic

The application will run as a service and perform following actions at startup and during its execution.

1. Load application configuration that handles different aspect of application
2. Start RMI server for CLI features
3. Load supported node types
4. Load node configuration and build runtime configuration
5. Start scheduler and schedule jobs according to configuration
6. Once job is triggered, following actions will be taken:
    a. Discover data from remote system
    b. Filter based on node configuration and history
    c. Transfer data
    d. Do housekeeping on node if applicable
    e. Update history with new changes

2.3 Directory Structure

Following directory structure will exist once the customization is installed. The directories. Directories like **`history`** and **`logs`** will be automatically created on paths specified in application configuration:

```
SIM/
├── bin
├── etc
├── lib
├── extlib
├── nodetypes
├── policy
└── protocol
```

3 Installation

3.1 Pre-requisites

Before this installation can be started the following pre-requisites must be fulfilled:

- Java version 1.8 installed
- `root` and `dcuser` access to server

3.2 Installation

SIM, for now, provides a simplified procedure for installation

1. Login to ENIQ Engine as `dcuser` user.
2. Transfer the installation package `tar` file to targeted server. Move to the directory where it should be installed.

   The preferred and recommended place to install SIM is `/eniq/sw/platform/`.

3. Extract the package

   ```
   $ tar -xvf <package_name>.tar.gz
   ```

   Once the package is extracted, there will be a new directory created by name `SIM` containing all require scripts and libraries for application.

4. Update permissions for scripts

   ```
   $ chmod 755 SIM/bin/*.sh
   ```

5. Change to `root` user for installation and setup of SIM service
6. To setup SIM as server, execute the install script provided in `bin` directory

   ```
   $ bin/install.sh
   ```

   The script will setup the service that will execute SIM as `dcuser`.

3.3 SIM service

SIM application is executed as a service and can be controlled by `service` command. Service can be managed as `root` user.

To start service:

```
service sim start
```

To stop service:

```
service sim stop
```

To check service status:

```
service sim status
```

3.4 Application Setup

The application provides helper scripts and configuration files for execution of the application.

3.4.1 Configuration

The application uses a configuration file in JSON format for setting up various parameters. Detailed information on each parameter used in configuration file is provided in chapter 4.

There is also nodes configuration file which is a CSV formatted file. This file should be setup with list of nodes from which data would be collected. More information on this file is provided in chapter 4.

3.4.2 start.sh

The `start.sh` script, present in `bin` directory can be used to start the application with necessary paths and properties set. The script facilitates in passing argument to the application as well. Following arguments are supported by the application:

Argument	Description
`--config`	Provide absolute path of configuration file to be used. The argument should be used in `--config=<path>` format. **Default:** If no argument is provided, the config `etc/config.json` will be used

Examples:

```
$ bin/start.sh --config=/opt/SIM/etc/custom.json
```

The scrip is used automatically when SIM is setup as service.

This script should not be used if SIM is started by service. Use service commands to start SIM.

3.4.3 stop.sh

The **stop.sh** script, present in the **bin** directory can be used to stop the application when required.

The script is used automatically when SIM is setup as service.

This script should not be used if SIM is started by service. It can cause the service to start the application again in background. Use service commands to stop SIM.

3.4.4 cli.sh

The **cli.sh** script can be used to start CLI client for SIM that will connect with SIM server and provide some information related to current configurations as well as adding new node types during application's execution.

The script facilitates in passing argument for setting up the user. Following arguments are supported by the application:

Argument	Description
--config	Provide absolute path of configuration file to be used. The argument should be used in **--config=<path>** format. **Default:** If no argument is provided, the config **etc/config.json** will be used

Example:

```
$ $ bin/cli.sh --config=/opt/SIM/etc/custom.json
```

Once started, the cli will present some options:

```
--------- Reading configuration ---------
Configuration read

--------- Connecting with server ---------
Connected with SIM_RMI

--------- Select an option to execute ---------
1- Print jobs
2- Install node type
3- Reload NE configuration
x- Exit
Choice:
```

3.4.4.1 Print Jobs

The first option will print current jobs that are scheduled. A job is created for each node that is configured for collection of data. The option will list all jobs along with information on when and how they will be executed and some execution statistics. The information is valid from since SIM is started or since last update of nodes configuration:

```
--------- Listing scheduled jobs ---------

--------- Job name: AIR4_Subnet2 ---------
Expression:       0 10,25,40,55 * ? * * *
Summary:          at 10, 25, 40 and 55 minutes
Next execution:   2022-01-17 15:10:00

Statistics:

        Total Job Executions    = 2675

        Last Tasks Executed     = 147

        Last Successful Tasks   = 147

        Last Failed Tasks       = 0

        Last Job Execution Time = 10.882 sec

        Total Tasks Executed    = 567363

        Total Successful Tasks  = 567363
        Total Failed Tasks      = 0
```

3.4.4.2 Install node type

This option allows for installation of new node type without restart of server. Node type defines a how transfer of data from a node will be treated and is a required parameter in node configuration. The option requires absolute path to node type jar file delivered by developers.

```
--------- Adding new node type ---------
Enter absolute path to jar file: /var/AIR.jar
Jar file: /var/AIR.jar
Installing node type
Node type added
Copying AIR.jar to node type dir: /opt/SIM/nodetypes
File copied
```

Once a path is provided, the jar is copied to node types directory, so it is available from next start of SIM. A request is also sent to SIM to install it and make it available for us.

3.4.4.3 Reload NE configuration

This option allows for the NE configuration to be reloaded and applied while SIM is running. NE configuration can be changed whenever required but it is only applied when reloaded through this option.

If new NEs are added to configuration, they will be scheduled according to configuration. If any NE is removed or commented from configuration, its schedule will be removed as well. If the job was executing while the configuration is applied, it will not take effect until the job starts the next time.

4 Configuration

SIM application consists of a few configuration files. The files can be placed anywhere accessible but by default they can be found in `SIM/etc/` directory.

Make sure all configuration changes are done as **`dcuser`**.

4.1 Application Config

The application configurations, by default, are save in `config.json` file. This file contains setup parameters that are loaded by application at startup. If any parameter is changed, the application should be restarted again to load the configuration again.

The configuration file is in JSON format in following format:

```
{
  "paths": {
    "nodeTypes": "",
    "neConfig": "",
    "historyDb": ""
  },
  "threadPool": {
    "minSize": 1,
    "maxSize": 100,
    "idleTimeout": 900,
    "awaitTermination": 15
  },
  "rmi": {
    "serverName": "SIM_RMI",
    "registryPort": 1879
  },
  "jobs": {
    "batchSize": 1000,
    "windDownTime": 45
  },
}
```

Supported parameters can be set with following values:

Name	Allowed Value	Description
`nodeTypes`	Valid dir path	Path the directory containing supported node type jar files. **Default:** If not path is provided, the application will look for node types in `nodetypes` directory inside application directory
`neConfig`	Valid file path	Path to node config file.

		Default: If not path is provided, the application will look for **`nodes.config`** file in applications **`etc`** directory
`historyDb`	Valid dir path	Path to directory where history or backlog information will be saved for each configured node. **Default:** If not path is provided, the application will use **`history`** directory inside application directory
`minSize`	Valid positive number	Number of minimum threads that will always be available for handling jobs. It is recommended to use a smaller number to not use resources when not required **Default:** 1
`maxSize`	Valid positive number	Maximum number of threads that will be created to handle the data transfer request for the configured nodes. It is recommended to not use extremely high number as it can negatively impact performance. As general rule, consider 1 thread for each node configured **Default:** 100
`idleTimeout`	Valid positive number	Time in seconds after which idle threads will timeout from thread pool. The thread pool will always keep **`minSize`** threads active even after timeout. **Default:** 900 seconds
`awaitTermination`	Valid positive number	Time to wait after shutdown call to the thread pool has been made. This give chance for current tasks to complete if they are near completion. **Default:** 15 seconds
`serverName`	Valid string	RMI server name which will be used to identify the server once started. This is required by CLI to connect correctly with the application. **Default:** SIM_RMI
`registryPort`	Valid available port	RMI port where application server will be registered. This is required by CLI to find the server and connect with it.

		Default: 1879
batchSize	Valid positive number	Number of files that can be scheduled at a time for a node. In cases where a node has many files to download during a ROP period, it can take all resource. This allows for processing files in batches for a node allowing other nodes to schedule download operation on thread pool. It is recommended to not provide a very small size as it can adversely affect performance. A very large batch size can also result in on node taking all thread pool for longer period **Default:** 1000
windDownTime	Valid positive number	Time in seconds until start of next ROP for a node to start procedure to stop current job. In case of long running job, backlog recovery for example, the job can overrun into next ROP period. This allows for SIM to initiate stopping the job in controlled way so it can be started again during next ROP period. Any files that were left for download will be considered in next ROP period. This allows for SIM to prioritize newer files during next ROP period while continuing with older files **Default:** 45 seconds

4.2 Node configuration

The node, or network element, configuration file contains configuration related to data collection from nodes. The configuration is a comma separated file, CSV, with header containing each configuration option that will be defined for each node. The first line should always be the header and should not be repeated. Any line starting with **#** will be considered as a comment and ignored.

Any configuration that is supported by a node type can be added to the node configuration file to use instead of default value provided by node types. However, following are mandatory configuration that must be present:

Header	Allowed Value	Description
name	Valid string	Name of the node identified in application.

IPAddress	Valid IP	IP address of the target node
pluginNames	Valid string	Must be one of the support node types by the application.
uniqueID	Valid string	Use to identify nodes with same name but different configuration or data collection properties. **name** and **uniqueID** are combined during execution to form a unique identified for node for representation of jobs **Default:** Empty string
rop	5, 15, 30, 60	Intervals at which data will be collected from node. **Default:** 15
Offset	0-60	Additional minutes that should be added to ROP period for data collection from the node. This is provided if data is generated on node sometime after ROP period. **Default:** 0
deleteAction	Empty string, 'IMMEDIATE', 'STORAGE_DAYS', or 'MOVE'	Delete action that will be performed if data files from node are considered for cleanup. An empty string means data files on node will not be cleaned up. IMMEDIATE will remove the file or node as soon as its downloaded STORAGE_DAYS works with **storageDays** parameter to remove file from node after defined number of days MOVE work with **moveTo** parameter to move file to another location on node after it is successfully downloaded **Default:** empty string
storageDays	Valid integer	Number of days for which the data files on nodes will be retained before considered for cleanup. This is part of the housekeeping function and work with the **deleteAction** parameter. Value of **0** means that data files on nodes will not be cleaned up even if **deleteAction** is set. **Default:** 0

moveTo	Valid path	Works with **deleteAction** parameter to move file to specified location on node after it is download **Default:** /var/tmp/
destinationDirectory	Path to dir	Path to directory where data files will be stored once transferred from nodes. This must be a valid path

In addition to above parameters, each node type can have additional parameters, some of which can be mandatory and must be added to node configuration file when using that node type. Each configuration that node types can provided, can be added to this configuration file, and used.

Each node configured must have a node type. Further information on node types is detailed in chapter 5.

5 Node Types

Node types, in SIM, defines configuration defaults that are required to handle file transfer from the nodes of that type. Each option that node types define can be overridden with specific value in the node's configuration file [4.2]. It should be noted that each configuration option provided by a node type must also be known to SIM. The supported options are listed in 5.1.

Services organization can provide new node types that can be installed in SIM following the procedure detailed in 3.4.4.2.

5.1 Sftp Node

A default node type, **`SftpNode`**, is already supported by SIM and is part of application. It defines all known options that are supported by SIM for SFTP operation for now. Additional node types can derive from **`SftpNode`** and can change the defaults with valid values to further modify how a node will be handled by SIM

`SftpNode` node type supports following configuration:

Name	Required	Default	Description
`sftpUserName`	Yes	None	Defines SFTP user that is used to login to node for file transfer
`sftpKeyFile`	No	Empty string	Path to key file on local system that can be used to login to node using the name provided by **`sftpUserName`** property
`sftpUserPass`	No	Empty string	Plain text password for user used to SFTP to node. **It is not recommended** to provide plain text password in node config file for nodes, however the option is still provided. The recommended solution is either to use key files or setup public key on nodes using **`ssh-copy-id`** utility. Make sure to verify SSH before configuring node on SIM to make sure it's not asking for password.
`sftpPort`	No	22	Valid SFTP port for the node.

sftpRetries	No	0	Number of SFTP operation retries to do before failing the operation.
remoteDirectory	Yes	None	Base directory on node where to look for files
namePattern	No	.*	Regular expression used to filter files on node
maxSftpConnections	No	5	Maximum number of parallel connections to create towards node for file transfer. This configuration must be done according to SSH demon configuration on node side

5.2 Sftp Appender

A default node type, **SftpAppender**, is already supported by SIM and is part of the application. It defines all known options that are supported by SIM for SFTP [5.1] as well as append operations. Additional node types can derive from **SftpAppender** and can change the defaults with valid values to further modify how a node will be handled by SIM.

SftpAppender is a generic node type that can support any stream of data. It is not mandatory to have a CSV format or any specific format.

SftpAppender node type supports following configuration:

Name	Required	Default	Description
appendToFile	No	True	Inform SIM that file appending is enabled for a matching file for certain node
appendRetries	No	3	Define how many retries should SIM perform towards a file on node before it is considered as an inactive file. An inactive file is one where no data in appended during a ROP period
finishDelimiter	No	\n	Define the delimiter considered as end of valid entry in node file. If there is no delimiter provided, all data that is available for reading since last update will be transferred to local files

5.3 CSV Appender

CsvAppender is derived from **SftpAppender** and defines all known options that are supported by SIM for Sftp Appender [5.2] as well as CSV append operations. Additional node types can derive from **CsvAppender** and can change the defaults with valid values to further modify how a node will be handled by SIM

CsvAppender node type supports following configuration:

Name	Required	Default	Description
appendToCsv	No	True	Informs SIM that CSV file appending is enabled for the node
hasHeader	No	True	Define if the file on node can have header. True will mean that first line of file will be considered as header. If value is set to false, then no header will be considered
csvDelimiter	No	,	Delimiter used to separate each value in a line
csvDateHeaderIndex	No	0	Identifies the column index containing header information. Date is used to identify the ROP period an entry/line belongs to
csvDateHeaderFormat	No	yyyy-MM-dd-HHmm	Date time format of date identified in a CSV line

6 Logging

All operations in application are logged in files. Log configuration, creation, and housekeeping is handled by **log4j** library.

6.1 Configuration

The configuration file for controlling logs is present at **SIM/etc/log4j.xml** and follows standard **log4j** configuration option.

6.1.1 Log Level

The amount of information logged depends on log level used. In recommended log level, **INFO**, it prints the operations completed. **DEBUG** level log should give more detailed information about the operations i.e., steps performed in a certain operation. If the highest log level is used, **TRACE**, then it also contents of responses and how things are compared. This is only meant for troubleshooting and has performance impact, as more information is written to disk more often, so do not leave the logging configured like this for normal operation.

6.2 Appenders

By default, the application is configured to use 3 appenders. In log4j appenders are responsible to sending logging events to destination.

6.2.1 Console

Console appender is used to stream log events to system console. The configuration provides a pattern in which information will be displayed. The amount of information displayed depends on logging level used for console output. By default, console logging is disabled, set to **OFF**. The option is provided for single executions or troubleshooting purposes only and is not supposed to be enabled during normal execution.

6.2.2 INFO_FILE

This file appender can be used to log all application activities. By default, the logs are sent to **SIM/logs/SIM_log.log** file. The default log level is **INFO**. For using different log level, use the configuration and suggestion detailed in [6.1.1].

6.2.3 ERROR_FILE

This file appender is used to log any errors that occur during execution of application. By default, the logs are sent to **`SIM/logs/SIM_error.log`** file. The default log level is **`ERROR`**.

Note that same errors will be logged to INFO_FILE as well. This specific appender is used to quickly check for errors rather than searching through INFO_FILE. Additionally, if higher log level is used for INFO_FILE, it can mean that files are rotated very quickly, and important error messages may be lost. In normal operations it is not expected to have many error messages so these logs should rotate after long times. In abnormal scenarios i.e., connectivity issues, there may be lot more errors logged.

6.3 Disk Space

The amount of disk space used depends on configuration of file rolling strategy and maximum file size and the number of files appenders. By default, for each file appender these values are set to 3 rolling files with each using maximum 10 MB. This means that when log file reaches 10 MB size, it will be rolled over and no more than 3 log files are kept for that specific appender. This totals to maximum 30 MB of disk space used by one appender.

In default configuration there are 2 file appenders specified. This will total to maximum of 60MB of disk space used in default configuration.

6.4 Housekeeping

Log files generated by application are automatically cleaned up according to log4j configuration.

7 History

For each node configured, there is a history file maintained in **historyDb** directory. History file maintains an image of data files on node and is updated as files are transferred or cleaned up on nodes. This allows SIM to create backlog in case it is started first time/restarted after a long pause, or ROPs were missed due to connectivity issue because of node or network outages. It must be noted that SIM can only transfer data that is available on node. If the node has cleaned up the data before SIM was started or had the opportunity to connect with the node, that data may be lost unless recovered on node side or manually transferred.

One history file is created for each node configured and the file is named using combination of **name** and **unqiueID** parameter option. Following JSON format is used to save history:

```
{
  "total": 1,
  "files": {
    "/path/20220117/sample.xml.gz": {
      "localPath": "/eniq/data/pmdata/staging/sample.xml.gz",
      "remoteMTime": 1642028000
    }
  }
}
```

The history files are maintained by application and must not be manually edited. If files are removed or changed outside of application, it can have undesired effects and can end up in error in collection, recollection of all files, or missed collection of files.

The first piece of information is total number of files maintained in the history, followed by all files that have been transferred.

Each file entry has absolute remote file path as key. The value contains local file path and remote modified time of the file

Parameter	Description
Total	Total number of files maintained in history
files	Contains list of all remote files processed by SIM
localPath	Absolute path on local system where file was downloaded
remoteMTime	Modified time of remote file

8 Terminology

8.1 Acronyms and Abbreviations

The acronyms and abbreviations used in this document are explained below.

CLI	Command Line Interface
CSV	Comma Separated Values
CU	Customer Unit
JSON	JavaScript Object Notation
RMI	Remote Method Invocation
ROP	Report Output Period
SFTP	Secure File Transfer Protocol
SIM	Simple Mediation
SSH	Secure Shell