Telephony Capture Service

Software Requirements Document



Table of Contents

1 Introduction 3

1.1 Purpose 3

1.2 Definitions 3

1.3 References 3

1.4 Open Issues 4

2 Overall Description 5

2.1 Product Perspective 5

2.1.1 External Interfaces 5

2.1.2 Internal Interfaces 5

2.1.3 User Interface 5

2.2 Project Deliverables 6

2.3 Technical Constraints 6

2.4 User Characteristics 6

3 Specific Requirements 7

3.1 TCS Requirements 7

3.2 Mangle SMDR Files 7

3.3 Test Tool Requirements 8

3.3.1 Telephone Exchange Simulator 8

3.3.2 Telecom Management System Simulator 9

3.4 Deployment Requirements 9

3.5 Test Requirements 9

3.6 Performance Requirements 10

# Introduction

## Purpose

This document details the software requirements for a Telephony Capture Service (TSC) that receives Station Messaging Detail Records (SMDR) from a telephone exchange and stores them to a database. The TCS is destined to replace an existing Telecom Management System (TMS), although a transition phase is anticipated in that the TCS can be configured to forward on all incoming data to the legacy TMS.

Other support utilities included in this project:

* A routine that ingests datasets of SMDR records and creates corresponding datasets containing otherwise identical information, except that all phone numbers are mangled (scrambled) in the copies. This utility provides a means to create versions of the SMDR data that can be released for off-site usage.
* Other utilities for testing:
  + A real-time telephony simulator that forwards a realistic stream of data to the TCS. Note: during the first installation this same utility is also used to load the historical set of SMDR data into the database.
  + A TMS simulator (accepts data from the TCS).

## Definitions

**SMDR**Station Messaging Detail Record. Its complete definition can be found in Appendix A.

**TMS**  
Telecom Management System

**TCS**  
Telephony Capture Service

## References

1. [SMDR Fields IPO 9.1.4 - required fields.docx](SMDR%20Fields%20IPO%209.1.4%20-%20required%20fields.docx)
2. [Telephony Capture Service Test Management Plan](TCS Test Management Plan.docx)

## Open Issues

* Investigate a means to safely identify duplicated SMDR data. This is required should a stream of SMDR messages need to be replayed.

# Overall Description

## Product Perspective

### External Interfaces

The TCS has 3 external interfaces, one is an input interface and two are outputs:

1. An input stream of SMDR messages from a telephone exchange. Note: this input stream also provides other unrelated messages, other than SMDR messages, hence, one of the TCS’s responsibilities is to filter-in only SMDR messages and exclude all others.
2. An output interface to a legacy Telecom Management System (TMS). This output is optionally active.
3. An output interface to a relational database table whose columns parallel the CSV field content of SMDR messages.

### Internal Interfaces

The TCS receives SMDR messages from the telephone exchange and must direct them both to the TMS and the database, but either or both of these may not be available and hence data destined for these output interfaces may be queued until such time that the TMS and / or the database are once again available. Further, queued data must be persisted should the TCS itself fail or is otherwise unavailable. The selected queuing service, [RabbitMQ](https://www.rabbitmq.com/), provides the services that are required.

### User Interface

The project’s User Interface components are minimal. These are:

1. A means to both view and edit deployment characteristics. This involves a manual text editor of the so-called Dockerfile that will be provided as part of this project. A change to the environment in which the TCS is running may require a change to the Dockerfile (using a simple text editor).
2. A means to view the number of items in the two RabbitMQ queues (should one or both of these queues start growing, this is indicative of a problem with the TMS and / or the database).
3. A means to edit various database configuration parameters and tables as required and according to project documentation (which is a project deliverable).

## Project Deliverables

All of the following to be delivered via GitHub:

* NodeJS source code (written in TypeScript)
* Executable JavaScript code (as transpiled by TypeScript)
* Projects documents: The present document, the Test Management Plan, and any diagram files that have been generated to support technical or user documentation.
* Application and deployment documentation.

## Technical Constraints

* Programming language: [TypeScript](https://www.typescriptlang.org/) 2.x
* Server Platform: [NodeJS](https://nodejs.org/en/) 6.9.x
* Database [PostgreSQL](https://www.postgresql.org/): 10.x
* Queue Message Broker: [RabbitMQ](https://www.rabbitmq.com/): 3.6.x
* Delivery Container: [Docker](https://www.docker.com/) 1.x

## User Characteristics

The users of the TCS are technical personnel who are familiar with the environment within which the TCS runs.

# Specific Requirements

## TCS Requirements

In parallel, the TCS shall:

* Receive TCP Segment Data from the telephone exchange and optionally deliver this data unmodified to a persisted queue, TMS\_QUEUE (as supported by RabbitMQ).

Note: If the Telecom Management System is eventually deemed unnecessary, then the TCS flow to the TMS\_QUEUE can be disabled.

* The TCS shall receive data from the TMS\_QUEUE and forward it on to the TMS (i.e. this data once again becomes TCP Segment Data).
* The TCS shall receive TCP Segment Data from the telephone exchange and isolates messages found between the data patterns (in hex) “00 02 00 00 00 00” and “0a 0d” (carriage return, line feed) and delivers such messages unmodified to two queues:1) DATABASE\_QUEUE and 2) LOG\_QUEUE.
* The TCS shall receive SMDR messages from DATABASE\_QUEUE, parses them into their respective CSV fields (SMDR messages are in a CSV format) and stores them into a database table.
* The TCS shall purge the LOG\_QUEUE, so as to ensure a queue size of between 3000 and 3500 SMRD Messages (note: LOG\_QUEUE is purely provided for contingency purposes and does not contribute to application).

## Mangle SMDR Files

A mangling tool shall be provided which supports the following requirements:

* The tool shall ingest the content of input files (of the form **rwyymmdd.00<d>**) containing SMDR records and creates corresponding output files (of the form **rwyymmdd.00<d+1>)** also containing SMDR records.
* For each SMDR record found in the input file a corresponding SMDR record shall be created in the output file, where the output record is identical, except that, where a phone number is unknown to the TCS, the last 4 digits of each input phone number shall be replaced with a random selection of 4 digits.

*Note: Some phone numbers are ‘known’ in the sense that the source of the call is from a phone that has been specifically installed to support the application. These numbers are not associated with particular individuals and hence privacy is not a concern. Mangling of such numbers is not required nor desirable.*

* The mangling shall be consistent throughout an input dataset, even crossing file boundaries. Example: if the number 6049424321 is mangled to 6049421234, then wherever the number 6049424321 is found in the dataset, then it will consistently be replaced with 6049421234 (even if the dataset contains SMDR records for multiple months).

*Notes:*

1. *Phone number mangling is done to ensure phone number privacy. The mangled output files can and will be used during development phases of this and other projects.*
2. *Currently the organization is using a single telephone exchange. An SMDR file with extension***.001** *indicates that its contents are genuine phone numbers.*
3. *Hence, SMDR files with extensions* **.002,. 003,** *etc*.*contain* only *mangled phone numbers.*
4. *It may be useful to think of SMDR files which have an extension* **.00x** *(x > 1), as being sourced from a virtual telephone exchange.*
5. *In principle, it is only necessary to mangle an SMDR .***001** *file once (to a 0.002 file), yet the software shall be able to ‘re-mangle’ .***002** *files to .***003** *files, .***003** *files to .***004** *files, etc. This capability is provided is required in order to allow for off-site testing of the mangling software itself.*

## Test Tool Requirements

A number of related test tools are required to support various stages of testing. The requirements for these tools are discussed next.

### Telephone Exchange Simulator

The Telephone Exchange Simulator will primarily be used to test the TCS, however, it will have one other import role: the actual ingestion into the database of all legacy SMDR records.

The TXS works as follows. It accepts the following inputs:

1. A input directory of SMDR files (it will accept files names of the following form: rwhhmmdd.00<d> (and ignore all other files).
2. An IP address of the platform on which the TCS is running; and
3. The Port number on which the TCS is listening.

If any of the above parameters are found to be missing or invalid (e.g. providing an illegal IP address), then the TXS shall abort.

### Telecom Management System Simulator

The TMS Simulator listens for a flow of messages from the TCS. It functionality is minimal and is largely provided to server as a data sink for the TCS during testing. Nevertheless, it will parse the incoming flow of messages looking for valid SMDR messages and, when found, shall write them to the system console.

## Database Requirements

The prime focus of the TCS is to capture telephony data and record this data to a database. This database in question is the first such to record operational data on a routine basis and lays the groundwork for further development projects. As such, it is important to define a stable, reliable, and recoverable environment for the database to exist. This section provides the database-specific requirements. The discussion invariably takes on the language of the selected database, Postgres.

## Deployment Requirements

The TCS and sundry is delivered via GitHub. The deployment vehicle is Docker and related tools. This sub-section articulates the deployment requirements:

* The TCS functionality shall be delivered as Docker containers (microservices). This includes both the queuing service, RabbitMQ, and the database, Postgres, as well as the project-specific microservices.
* The consequence is that no external software, other than Docker itself, needs installation.

Note that this document does not take a position as the nature of the Docker installation, that is, installed within a Virtual Machine or installed on bare metal.

## Test Requirements

Testing of the TCS is done on two levels:

**Unit Testing**  
Using the Mocha Test Framework, suitable Test Cases will be crafted that will exercise the internal machinations of the TCS. Code coverage is required.

**Test Management**  
A separate Test Management Plan to be composed. It will describe in detail the various Test Cases to be carried successfully in order to ensure that the TCS is painlessly introduced into the operational environment.

This Test Management Plan shall also identify a “minimum suite of Test Cases” that will need to be successfully executed before subsequent upgrade versions of the TCS can be released into operations.

## Performance Requirements

The minimum performance of the TCS is the following (note: these performance requirements are not demanding):

1. Able to ingest and deliver to the TMS 1000 SMRD messages per hour for a sustained period of 3 hours.
2. Able to ingest and deliver to the database 1000 SMRD messages per hour for a sustained period of 3 hours.
3. Without message loss, be able to ingest and deliver to the TMS a traffic burst of 1000 SMRD messages in 10 seconds.
4. Without message loss, be able to ingest and deliver to database a traffic burst of 1000 SMRD messages in 10 seconds.