# Technical Reference Manual

## GIT for Industry

| | |
|---|---|
| **Version:** | **4.3.0** |
| **Date:** | **28th January 2021** |
| **Author:** | **Smart DCC Ltd.** |
| **Classification:** | **DCC Public** |

# Document Control

### Revision History

| Revision Date | Summary of Changes | Changes Marked | Version Number |
|---|---|---|---|
| 28/10/2015 | First version. | N/A | 1.0RC4 |
| 27/11/2015 | Update | N/A | 1.0RC5 |
| 03/08/2016 | Update for enhanced version | N/A | 1.3.2E |
| 08/08/2017 | Update | N/A | 1.3.8E |
| 25/08/2017 | Update | N/A | 2.0.0 RC1 |
| 30/11/2017 | Update | N/A | 2.0.0 RC2 |
| 09/02/2017 | Update | N/A | 2.0.0 |
| 27/03/2018 | Update | N/A | 2.0.1 |
| 30/04/2018 | Update | N/A | 2.0.2 |
| 28/05/2018 | Update | N/A | 2.0.3 |
| 05/07/2018 | Update | N/A | 2.1.0 |
| 09/10/2018 | Update | N/A | 2.1.1 |
| 16/01/2019 | Update | N/A | 2.1.2 |
| 06/03/2019 | Update | N/A | 2.1.3 |
| 24/04/2019 | Update | N/A | 2.1.4 |
| 17/06/2019 | Update | N/A | 2.1.5 |
| 06/09/2019 | Update | N/A | 2.1.6 |
| 13/09/2019 | Update | N/A | 3.0.0 |
| 25/10/2019 | Update | N/A | 3.0.1 |
| 14/11/2019 | Update | N/A | 3.0.2 |
| 09/01/2020 | Update | N/A | 3.1.0 |
| 31/01/2020 | Update | N/A | 3.1.1 |
| 13/03/2020 | Update | N/A | 3.2.0 |
| 23/04/2020 | Update | N/A | 3.2.1 |
| 08/05/2020 | Update | N/A | 3.3.0 |
| 19/06/2020 | Update | N/A | 4.0.0 |
| 26/06/2020 | Update | N/A | 4.0.1 |
| 09/07/2020 | Update | N/A | 4.1.0 |
| 24/07/2020 | Update | N/A | 4.2.0 |
| 31/07/2020 | Update | N/A | 4.2.1 |

| Revision Date | Summary of Changes | Changes Marked | Version Number |
|---|---|---|---|
| 12/08/2020 | Update | N/A | 4.2.2 |
| 18/08/2020 | Update | N/A | 4.2.3 |
| 31/08/2020 | Update | N/A | 4.2.4 |
| 10/09/2020 | Update | N/A | 4.2.5 |
| 15/09/2020 | Update | N/A | 4.2.6 |
| 18/09/2020 | Update | N/A | 4.2.7 |
| 06/10/2020 | Update | N/A | 4.2.8 |
| 26/11/2020 | Update | N/A | 4.2.9 |
| 28/01/2021 | Update | N/A | 4.3.0 |

### Reviewers

| Name | Title / Responsibility | Release Date | Version Number |
|---|---|---|---|
| Gonçalo Gouveia | Critical Software Project Engineer | | 1.0RC5 |
| Karim Kanso | Critical Software Senior Engineer | | 1.3.2E |
| Dinis Paes | Critical Software Technical Manager | | 1.3.8E |
| Dinis Paes | Critical Software Technical Manager | | 2.0.0 RC1 |
| Dinis Paes | Critical Software Technical Manager | | 2.0.0 RC2 |
| Pedro Almeida | Critical Software GFI Team Leader | 09/02/2017 | 2.0.0 |
| Helder Martins | Critical Software GFI Team Member | 27/03/2018 | 2.0.1 |
| Helder Martins | Critical Software GFI Team Member | 30/04/2018 | 2.0.2 |
| Helder Martins | Critical Software GFI Team Member | 28/05/2018 | 2.0.3 |
| Helder Martins | Critical Software GFI Team Member | 05/07/2018 | 2.1.0 |
| Helder Martins | Critical Software GFI Team Member | 09/10/2018 | 2.1.1 |
| Helder Martins | Critical Software GFI Team Member | 16/01/2019 | 2.1.2 |
| Francisco Pinto | Critical Software GFI Team Leader | 06/03/2019 | 2.1.3 |
| Francisco Pinto | Critical Software GFI Team Leader | 24/04/2019 | 2.1.4 |
| Francisco Pinto | Critical Software GFI Team Leader | 17/06/2019 | 2.1.5 |
| Helder Martins | Critical Software GFI Team Member | 06/09/2019 | 2.1.6 |
| Helder Martins | Critical Software GFI Team Member | 13/09/2019 | 3.0.0 |
| Helder Martins | Critical Software GFI Team Member | 25/10/2019 | 3.0.1 |
| Francisco Pinto | Critical Software GFI Team Leader | 14/11/2019 | 3.0.2 |

| Name | Title / Responsibility | Release Date | Version Number |
|---|---|---|---|
| Helder Martins | Critical Software GFI Team Member | 09/01/2020 | 3.1.0 |
| Helder Martins | Critical Software GFI Team Member | 31/01/2020 | 3.1.1 |
| Nuno Besteiro | Critical Software GFI Team Member | 13/03/2020 | 3.2.0 |
| Sara Alves | Critical Software GFI Team Member | 23/04/2020 | 3.2.1 |
| Sara Alves | Critical Software GFI Team Member | 08/05/2020 | 3.3.0 |
| Nuno Besteiro | Critical Software GFI Team Member | 19/06/2020 | 4.0.0 |
| Nuno Besteiro | Critical Software GFI Team Member | 26/07/2020 | 4.0.1 |
| Nuno Besteiro | Critical Software GFI Team Member | 09/07/2020 | 4.1.0 |
| Helder Martins | Critical Software GFI Team Member | 24/07/2020 | 4.2.0 |
| Nuno Besteiro | Critical Software GFI Team Member | 31/07/2020 | 4.2.1 |
| Francisco Pinto | Critical Software GFI Team Leader | 12/08/2020 | 4.2.2 |
| Helder Martins | Critical Software GFI Team Member | 18/08/2020 | 4.2.3 |
| Helder Martins | Critical Software GFI Team Member | 31/08/2020 | 4.2.4 |
| Francisco Pinto | Critical Software GFI Team Leader | 10/09/2020 | 4.2.5 |
| Daniel Oliveira | Critical Software GFI Team Member | 15/09/2020 | 4.2.6 |
| Francisco Pinto | Critical Software GFI Team Leader | 18/09/2020 | 4.2.7 |
| Daniel Oliveira | Critical Software GFI Team Member | 06/10/2020 | 4.2.8 |
| Francisco Pinto | Critical Software GFI Team Leader | 26/11/2020 | 4.2.9 |
| Francisco Pinto | Critical Software GFI Team Leader | 28/01/2021 | 4.3.0 |

### Approvals

| Name | Signature | Title / Responsibility | Release Date | Version Number |
|---|---|---|---|---|
| António Alves | | Critical Software Technical Manager | 09/08/2016 | 1.3.2E |
| Dinis Paes | | Critical Software Technical Manager | 08/08/2017 | 1.3.8E |
| Dinis Paes | | Critical Software Technical Manager | 25/08/2017 | 2.0.0 RC1 |
| Dinis Paes | | Critical Software Technical Manager | 04/12/2017 | 2.0.0 RC2 |
| Pedro Almeida | | Critical Software GFI Team Leader | 09/02/2018 | 2.0.0 |
| Pedro Almeida | | Critical Software GFI Team Leader | 27/03/2018 | 2.0.1 |
| Pedro Almeida | | Critical Software GFI Team Leader | 30/04/2018 | 2.0.2 |
| Pedro Almeida | | Critical Software GFI Team Leader | 28/05/2018 | 2.0.3 |

| Name | Signature | Title / Responsibility | Release Date | Version Number |
|---|---|---|---|---|
| Pedro Almeida | | Critical Software GFI Team Leader | 05/07/2018 | 2.1.0 |
| Pedro Almeida | | Critical Software GFI Team Leader | 09/10/2018 | 2.1.1 |
| Pedro Almeida | | Critical Software GFI Team Leader | 16/01/2019 | 2.1.2 |
| Pedro Almeida | | Critical Software Technical Manager | 06/03/2019 | 2.1.3 |
| Pedro Almeida | | Critical Software Technical Manager | 24/04/2019 | 2.1.4 |
| Pedro Almeida | | Critical Software Technical Manager | 17/06/2019 | 2.1.5 |
| Pedro Almeida | | Critical Software Technical Manager | 06/09/2019 | 2.1.6 |
| Pedro Almeida | | Critical Software Technical Manager | 13/09/2019 | 3.0.0 |
| Pedro Almeida | | Critical Software Technical Manager | 25/10/2019 | 3.0.1 |
| Pedro Almeida | | Critical Software Technical Manager | 14/11/2019 | 3.0.2 |
| Pedro Almeida | | Critical Software Technical Manager | 09/01/2020 | 3.1.0 |
| Pedro Almeida | | Critical Software Technical Manager | 31/01/2020 | 3.1.1 |
| Pedro Almeida | | Critical Software Technical Manager | 13/03/2020 | 3.2.0 |
| Pedro Almeida | | Critical Software Technical Manager | 23/04/2020 | 3.2.1 |
| Pedro Almeida | | Critical Software Technical Manager | 08/05/2020 | 3.3.0 |
| Pedro Almeida | | Critical Software Technical Manager | 19/06/2020 | 4.0.0 |
| Pedro Almeida | | Critical Software Technical Manager | 26/06/2020 | 4.0.1 |
| Pedro Almeida | | Critical Software Technical Manager | 09/07/2020 | 4.1.0 |
| Pedro Almeida | | Critical Software Technical Manager | 24/07/2020 | 4.2.0 |
| Francisco Pinto | | Critical Software GFI Team Leader | 31/07/2020 | 4.2.1 |
| Pedro Almeida | | Critical Software Technical Manager | 12/08/2020 | 4.2.2 |
| Pedro Almeida | | Critical Software Technical Manager | 18/08/2020 | 4.2.3 |
| Pedro Almeida | | Critical Software Technical Manager | 31/08/2020 | 4.2.4 |
| Pedro Almeida | | Critical Software Technical Manager | 10/09/2020 | 4.2.5 |
| Pedro Almeida | | Critical Software Technical Manager | 15/09/2020 | 4.2.6 |
| Pedro Almeida | | Critical Software Technical Manager | 18/09/2020 | 4.2.7 |
| Pedro Almeida | | Critical Software Technical Manager | 06/10/2020 | 4.2.8 |
| Pedro Almeida | | Critical Software Technical Manager | 26/11/2020 | 4.2.9 |
| Pedro Almeida | | Critical Software Technical Manager | 28/01/2021 | 4.3.0 |

# Table of Contents


**1 Introduction ..... 10**

1.1 Objective ..... 10

1.2 Document structure ..... 10

**2 Overview ..... 10**

2.1 High Level Architecture ..... 10

2.1.1 VSIS-Core ..... 11

2.1.2 Simulation ..... 11

2.1.3 GFI Testing Tool ..... 11

2.1.4 Test Library ..... 12

2.1.5 Messages Database ..... 12

2.1.6 GFI-Testing-Tool GUI ..... 12

2.1.7 Communications Hub ..... 12

2.2 Reference Testbed ..... 13

2.3 Inputs and Outputs ..... 14

2.3.1 Configuration File ..... 15

2.3.2 Test Properties File ..... 15

2.3.3 Test Report Files ..... 15

2.3.4 Console Output ..... 15

2.3.5 Execution Log ..... 15

2.3.6 Communications Hub Configuration File ..... 16

2.4 Test Execution ..... 16

2.5 Results Analysis ..... 16

**3 Test Setup ..... 16**

3.1 Scenario Configuration File ..... 17

3.1.1 Console and Log Output ..... 17

3.1.2 Test Case Scheduling ..... 18

3.1.3 Result Manager ..... 19

3.1.4 Simulation Manager ..... 19

3.1.5 Emulators ..... 19

3.1.6 Client ..... 27

3.2 Test Properties File ..... 28

3.2.1 Report Information ..... 29

3.2.2 Resource Combiner ..... 30

3.2.3 Resource Pool ..... 30

3.2.4 Test Repetition ..... 31
**4 Communications Hub ..... 32**
4.1 Configuration File ..... 32
4.1.1 Interfaces Configuration ..... 32
4.1.2 HAN Configuration ..... 33
4.1.3 GPF configuration ..... 34
4.1.4 CHF configuration ..... 35
4.1.5 CHF Device Log ..... 37
4.2 Execution ..... 38
4.3 Logging ..... 41
4.4 Control Port ..... 41
4.4.1 Exit ..... 42
4.4.2 Get Key Table ..... 43
4.4.3 Add Device ..... 43
4.4.4 Remove Device ..... 43
4.4.5 UnJoin Device ..... 44
4.4.6 Get Stack Versions ..... 44
4.4.7 Get CH Time ..... 44
4.4.8 Sync CH Time with Operative System ..... 44
4.4.9 Set CH Time ..... 44
4.4.10 CH Soft Stop ..... 45
4.4.11 CH Hard Stop ..... 45
4.4.12 Get Zigbee network info ..... 45
4.4.13 Get CH device info ..... 45
**5 Test Execution ..... 46**
5.1 Command Line Execution ..... 46
5.2 Command Line Options ..... 47
5.2.1 Help ..... 47
5.2.2 Version ..... 48
5.2.3 Cryptography ..... 48
5.2.4 Offline Check ..... 49
5.2.5 Verbose Level ..... 49
5.2.6 Output Directory ..... 50
5.3 Test Report: XML ..... 50
5.3.1 Properties ..... 51
5.3.2 Overall Result ..... 51

5.3.3 Test Case ..... 52
5.4 Test Report: HTML ..... 55
5.4.1 Test Information ..... 55
5.4.2 Test Cases Results ..... 56
5.4.3 Test Cases Execution ..... 56
5.4.4 Test Case Results Summary ..... 58
5.5 Console Execution Output ..... 59
5.5.1 Time ..... 59
5.5.2 Area ..... 59
5.5.3 Level ..... 59
5.5.4 Message ..... 59
5.6 Execution Log File ..... 60
5.6.1 Sequence number ..... 60
**6 Use Case Creation/Upgrade ..... 60**
6.1 Process Overview ..... 61
6.2 Use Case Specification ..... 62
6.2.1 Command Payload ..... 62
6.2.2 Response Payload ..... 63
6.2.3 Alert Payload ..... 64
6.2.4 Use Case Test ..... 64
6.2.5 XML Assembling ..... 64
6.2.6 Use Case Configuration ..... 66
6.2.7 Custom Message Code ..... 67
6.2.8 Match Message Code and Message Type ..... 67
6.3 Use Case Coding ..... 68
6.3.1 Base Class Files ..... 69
6.3.2 Command Class Files ..... 69
6.3.3 Response Class Files ..... 69
6.3.4 Alert Class Files ..... 70
6.3.5 Test Case Class files ..... 70
6.3.6 Emulator Class ..... 70
6.3.7 Store Model ..... 70
6.3.8 Compilation ..... 71
6.3.9 Installation ..... 73
6.4 Use Case Execution ..... 74
6.4.1 Execution with GUI ..... 75

6.5 Use Case Validation ..... 75
**7 Adding New Test Cases to the GUI ..... 75**
7.1 Test Case Database File ..... 76
7.2 Manufacturer Test Cases File ..... 77
**8 Business Scenarios ..... 78**
**9 OTA Functionality ..... 79**
9.1 OTA Tool ..... 79
9.1.1 Build OTA Upgrade Image ..... 80
9.1.2 Verify OTA Upgrade Image ..... 81
9.1.3 Verify Manufacturer Image construction ..... 82
9.2 Send Image to Devices ..... 83
9.2.1 Send Image notify to devices by node Id ..... 83
9.2.2 Send Image notify to all devices by broadcast ..... 83
**10 Glossary ..... 83**

# 1 Introduction

## 1.1 Objective

This document is a Technical Reference Manual for the GFI testing tool intended to provide detailed technical information for standard and advanced user operation and configuration.

## 1.2 Document structure

Section 1 (Introduction) presents a general description of this document's contents.

Section 2 (Overview) presents a general description of the framework operation.

Section 3 (Test Setup) presents a detailed description of the Test setup process.

Section 4 (Communications Hub) presents information relating to the configuration and execution of the communicataions hub.

Section 5 (Test Execution) presents a detailed description of the Test execution process.

Section 6 (Use Case Creation) presents a detailed description of how to create a new Use Case. This section is minded for Advanced Users only.

Section 7 (Adding New Test Cases to the GUI) presents a detailed description of how to add new Test Cases to the GUI.

Section 8 (Business Scenarios) presents the location of Business Scenarios manuals and workspaces.

Section 10 (Glossary) presents a list of definitions and acronyms used throughout this document.

# 2 Overview

GIT for Industry (GFI) is a software tool, provided by Smart DCC, for anybody that wishes to check whether their interpretation of the Great Britain Specification Companion for smart meters (GBCS) is consistent with Smart DCC's. At the time of writing, GFI supports Use Cases for GBCS v4.0 over a ZigBee HAN. In addition to the library of Use Cases, GFI allows end users to create new, or extend existing GBCS Use Cases. This manual provides detailed technical information for advanced operation of the tool.

## 2.1 High Level Architecture

GFI is a testing tool and systems validation competence centre. In the core of the tool lays a message oriented infrastructure, where the message field is the most elementary entity. In a simplified overview, the tool executes elementary operations over message fields and messages, namely sets, checks, gets, sends and waits, among a few others. Therefore, a fundamental concept that should always be kept in mind is the concept of message and respective fields.

Messages and respective fields are defined in a database, automatically generated from message specifications. This database creates an abstraction layer between the engine of the testing tool (and the tests themselves) and the protocols and interfaces, through which messages are sent and received, leveraging a high level of decoupling between Tests, communication protocols and transmission medium.

To execute and produce the respective reports, modify and/or create Use Cases, a number of steps need to be taken. The following Sections provide detailed technical information for standard and advanced User operation. Also a top level architecture overview is provided in the next Section, describing each one of the main modules that form the testing tool.

Figure 1 provides a high level overview of the system's architecture. There are six main modules (shown in purple): GFI testing tool, VSIS Core, Communication Hub, Simulation, Test Library and Messages Database.

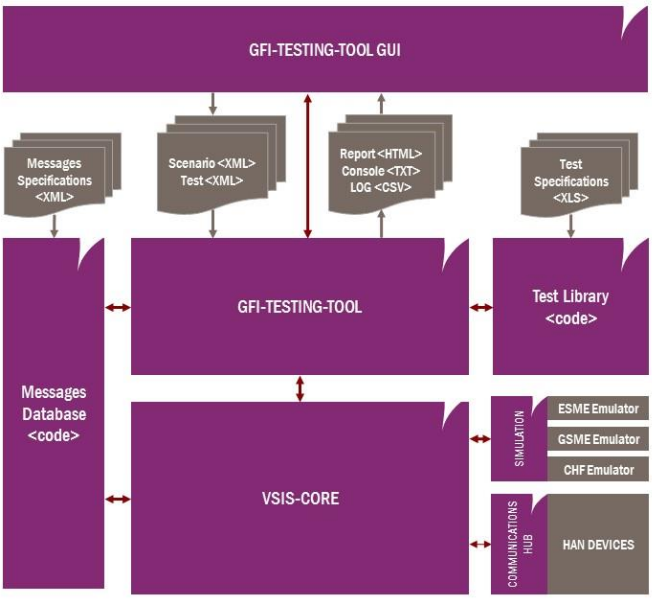


**Figure 1 – GFI high level architecture.**

### 2.1.1 VSIS-Core

This is the central element of the framework, provided as a library. This is where all the logic and functionalities of the testing tool are concentrated. This module is responsible for the creation of the test environment and all the necessary resources for the test execution:

- The execution of the test;
- Generating all the log information;
- Generating all the execution reports;

### 2.1.2 Simulation

The Simulation Manager is responsible for handling the routing of GBCS messages between the KRP and the Devices when the test is executed in an emulated scenario (with emulated Devices).

### 2.1.3 GFI Testing Tool

This is the highest layer of the test framework, responsible for establishing the connections between Core, Simulation Manager, Communications Hub and the Test Library. This tool may be invoked either by the GUI (which generates the commands to be executed) or directly from the command line.

### 2.1.4 Test Library

This is the library containing all the Use Cases specified by GBCS and available for the Tester to use in the construction of Tests.

### 2.1.5 Messages Database

This is a database containing all the messages that provide support for the execution of the Use Cases. For instance, this database contains the General Ciphering Message and General Signing Message as well as all the other messages specified by GBCS.

### 2.1.6 GFI-Testing-Tool GUI

The Testing Tool's Graphical User Interface, providing functionality for editing, executing and analysing user-implemented test procedures.

### 2.1.7 Communications Hub

Beginning with GFI version 1.3.2E, an implementation of a communications hub[1] is offered which acts as a gateway to a persistent ZigBee HAN where physical devices may be connected, and supports several CHF and GPF features including maintaining a whitelist of devices allowed in the HAN and mirroring of GSME data. Figure 2 shows the Communications Hub's architecture. Next is a brief description of each block.

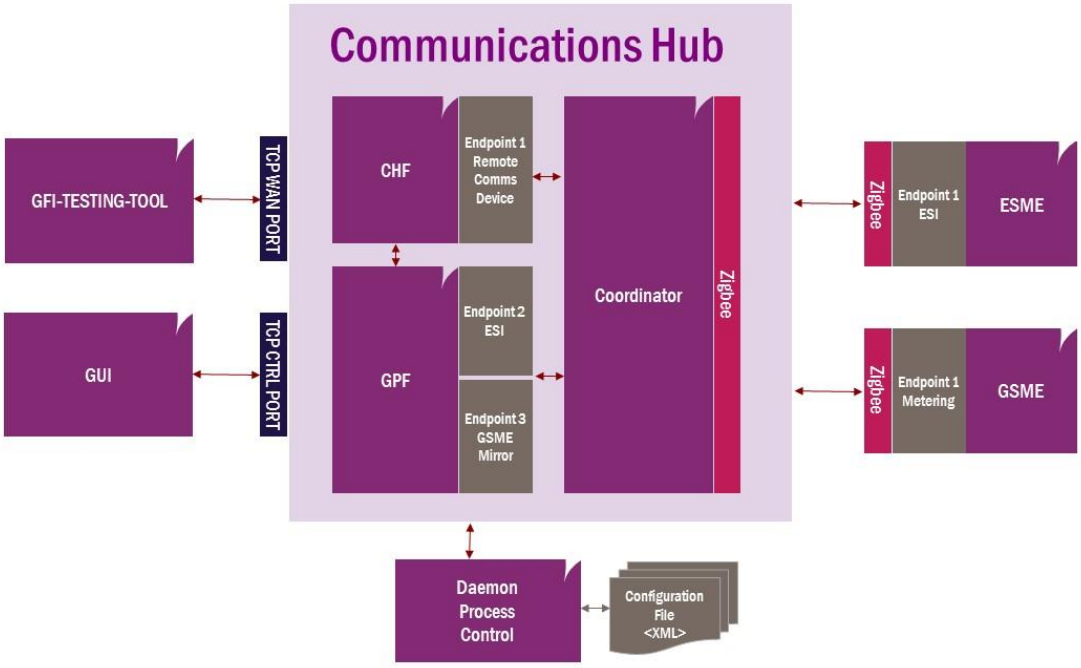


**Figure 2 – Communications Hub's Architecture**

[1] This should not be confused with the CHF/GPF emulators offered in the fully-emulated option. This module is the gateway to a real HAN (using physical devices) via a ZigBee adaptor.

### Daemon Process Control

The Communications Hub runs as a service. As such, and upon starting, it will run in the background waiting for requests. The Daemon Process Control is an ordinary script which provides an interface to the Communications Hub process, supporting the usual service commands such as start, stop, status and restart. To start the Communications Hub, a configuration file needs to be provided. The GUI provides a friendly user-interface to build the start-up configuration file and issueing the commands to control the communications Hub.

One consequence of being a service is the capability of persisting information between tests, as will be seen in later sections, something which cannot be achieved using full emulation, as state is not kept between tests in this latter case.

### CHF

The Communications Hub Function will service all test-cases in which it is the target device (e.g. CCS01) and validate and forward messages destined to other devices (e.g. GPF, ESME). As shown in Figure 2, the GFI-testing-tool uses a TCP port (referred to as WAN TCP port) to send/receive the GBCS messages. There is an additional TCP port used by the Communications Hub referred as the Control port, which can be used by any application to issue commands to the Communications Hub (it is used by the GUI to query the list of devices).

### GPF

The Gas Proxy Function's main role is to support a copy of GSME data. This copy is maintained using the ZSE mirroring and GBCS Tapping Off (TOM) mechanisms as described in GBCS. The GPF is then able to provide this data both to the WAN using the relevant GBCS use-cases and to the HAN using ZSE commands.

### Coordinator

The coordinator manages the interactions with the ZigBee stack and makes the functionality supported by the stack available to the CHF and GPF modules.

### Endpoints

The Communications Hub provides the following ZigBee endpoints:

- Remote Communications Device Endpoint: used for communication in the HAN.
- Gas ESI Endpoint: mirroring of GSME data and processing of Tapping Off commands.
- Gas Mirror Endpoint: mirroring of GSME data.

## 2.2 Reference Testbed

The GFI framework operates as a home area network (HAN) and emulates Remote Parties, ACB and CHF. It creates a ZigBee network and allows Devices to join. Through these connections it communicates with Devices in order to execute the Tests. In this communications, messages are exchanged using the GBCS protocol: commands are sent and the relevant Devices' responses and alerts are gathered to produce the Test Reports and to verify the Devices' conformance with the protocol. The reference Testbed is presented in Figure 3.

Note: The GFI testing tool only performs conformance tests against the GBCS v4.0 protocol, not functional tests. Although some minimal functional tests can be implemented and are in fact supported, that is not the purpose of the tool.

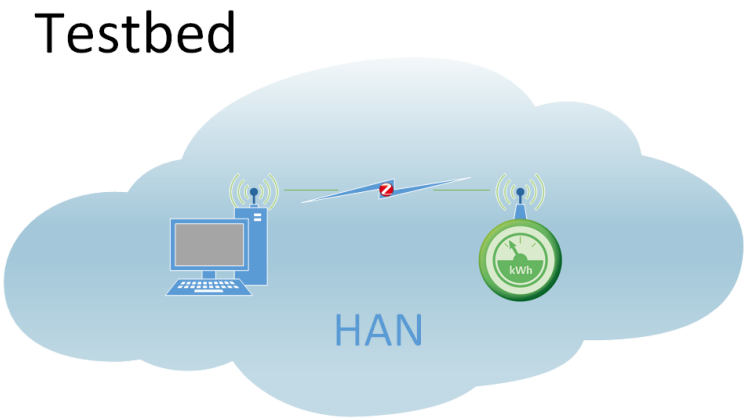


**Figure 3 – GFI Testbed diagram.**

## 2.3 Inputs and Outputs

Table 1 presents the inputs required and the outputs produced by the testing tool. A short description for each artefact is also presented.

| File | I/O | Type | Description |
|---|---|---|---|
| <scenario-file>.xml | Input | Configuration file | The scenario configuration file required by the tool specifies what equipment to use; e.g., the device configurations, keys and certificates. |
| <properties-file>.xml | Input | Test properties file | The Test properties file required by the tool specifies which Test Cases should be executed, the input data and the expected results for each Test Case. |
| <test-report>.xml | Output | Test report file | The raw execution report produced by the tool in XML format. It can be used by third-party tools to produce a customized report. |
| <test-report>.html | Output | Test report file | The final execution report produced by the tool in HTML format. Highlights all the relevant actions executed in the test as well as the results of each Test Case, Test Case iteration and Test Case step. |
| <execution-log>.csv | Output | Execution log file | The full execution output in comma-separated values format. Contains the full detail available from the tool's execution. Every single action (set, check, send, wait) performed by the tool is recorded in this file. |

| File | I/O | Type | Description |
|---|---|---|---|
| <ch_conf_*timestamp*>.xml | Input | Communications Hub configuration file | An XML configuration file which the CH loads upon booting up. Defines general settings, ZigBee radio configurations and may also add a set of devices to the CH's whitelist. |

**Table 1 – Input and Output artefacts.**

The following Sections contain a description for each one of these artefacts and the role they play in the system.

### 2.3.1 Configuration File

To setup a working environment for a Test, all the configurations should be defined in the scenario file <scenario-file>.xml. In this file, the settings for the produced outputs, test scheduler, emulators, equipment and codecs used may be tweaked to cope with environment needs. For instance, a Test can be executed in a scenario with real ESME/GSME devices or in a scenario with emulated devices. This configuration is defined in this file. A detailed description of this file and all of its components and options is presented in Section 3.1.

### 2.3.2 Test Properties File

The Test Cases that constitute the Test should be defined in the Test Properties file <test-properties>.xml. After the Test specific configurations at the top of the file, information regarding each Test Case (directly mapped to the SMETS Use Case) should be added, namely input parameters and expected outputs. Again, a detailed description of this file, organisation, input parameter definition and options is available in later Section 3.2.

### 2.3.3 Test Report Files

Upon the Test execution, two report files are generated: a raw XML with all the sets, checks, action prints and exchanged messages and a formatted HTML with a human readable appearance. Information on these files is detailed in Sections 5.3 and 5.4, respectively. These files may be considered the final outputs and contain the overall PASS/FAIL information as well as the detailed PASS/FAIL for each expected result.

### 2.3.4 Console Output

All the relevant actions executed in the Test are outputted to the console. The detail level of this execution output can be customized either by using a command line option or in the scenario configuration file <scenario-file>.xml. The higher the value chosen for the detail level, the more information will be displayed in the console. At the maximum detail level every bit of information available will be presented. Further information on this subject is accessible in later Section 5.5.

### 2.3.5 Execution Log

Like the console output, the tool also produces an execution log with a configurable level of detail defined in the Scenario Configuration file <scenario-file>.xml (see Section 3.1.1). The higher the value chosen for the detail level, the more information will be recorded in the log. It should always be the highest possible value (default) for the sake of record keeping. Further information on this subject is accessible in later Section 5.6.

### 2.3.6 Communications Hub Configuration File

To start the Communications Hub a configuration file needs to be provided. The GUI can be used to create and store several configurations, corresponding to different environments (e.g. different ZigBee radio settings and/or pre-configured whitelists). The GUI provides a user-friendly way of generating this file and issueing the commands to the CH, shielding the user from having to deal with the command line. The files generated feature a timestamp and will always be deleted upon shutting down the daemon. The process of generating a CH configuration file should ideally be the first step when the tool is operating in physical mode, and will be described in detail in Section 4.1.

## 2.4 Test Execution

For a Test to be executed the testing tool application should be invoked in a command line, using the Scenario Configuration file and the Test Properties file as parameters. In Figure 4 an example of the command line common call is provided.

```
gfi@gfi:/opt/gfi$ ./gfi-testing-tool conf/gfi-scenario-emulators.xml tests/ATG-TEST-ECS40.xml
```

**Figure 4 – GFI tool command line invocation example.**

If the test is against a physical meters, before executing the gfi-testing-tool it is required that the communications hub deamon is started. An example for starting the communications hub daemon is in Figure 5. More information about the communications hub can be found in Section 4

```
gfi@gfi:/opt/gfi$ ./bin/gfi-comms-hub conf/my-ch-configuration.xml
```

**Figure 5 – GFI communications hub command line invocation example.**

## 2.5 Results Analysis

After execution is complete a number of output artefacts (described in Section 2.3) are produced and the results analysis may be performed. On the console output a configurable execution information set is shown including the overall test result: PASS/FAIL.

An output log is generated with all the information of the Test execution. This log is customisable and all the information about its configurations and options can be found in Section 5.6 (Execution Log File)

Also, two report files are generated: a raw XML with all the sets, checks, actions, prints and exchanged messages and a formatted HTML with a human readable appearance. Information on these files is detailed in Sections 5.3 and 5.4 respectively. These files may be considered the final outputs and contain the overall PASS/FAIL information as well as the detailed PASS/FAIL for each expected result.

# 3 Test Setup

Before a Test can be executed, the input artefacts should be to be properly set, namely the Scenario Configuration file and the specific Test Properties file. Each input artefact is described in each of the following Sections. A general overview will be presented followed by a detailed description of each of its components.

## 3.1 Scenario Configuration File

This Section describes the Scenario Configuration file referred to as <scenario-file>.xml. This file contains all the configurations needed for the Test environment. It is one of the input parameters for the GFI testing tool and should always be present. In the following subsections a detailed description of all the components in this file is presented as well as all the configurable options available.

For a complete example of the scenario files, either use the GUI to generate them or look in the "/opt/gfi/conf" directory on the GFI platform. Within this directory are two example scenario files, one for the emulated scenario that is complete, and a template that can be used for creating physical scenarios.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE esme_scenario_configuration_physical_campaign >
<scenarioconfiguration>
  <testscenario>
    <default_output class="CAggregOutputter" level="200" name="OutputManager">
      <output class="CATGConsoleOutputter" level="4" name="ConsoleOutputter" />
      <output class="CATGBufferedFileOutputter" level="100" name="FileOutputter" />
    </default_output>

    <configuration_manager class="CDefaultConfigManager" name="ConfigManager">
      <item name="counters_persistency" value="true" />
      <item name="system_datetime" value="2015/01/01 00:00:00" />
    </configuration_manager>

    <scheduler class="CSchedulerTestCaseInterleaving" name="Scheduler" />

    <result_manager class="CGenericResultManager" name="ResultManager" />

    <equipment class="CSimulationManager" name="SimulationManager">
      <period>100</period>
    </equipment>

    <equipment class="CACBEmulator" name="ACBEmulator">
      <timeout_messages>3600000</timeout_messages>
    </equipment>

    <equipment class="CRemotePartyEmulator" name="RPEmulator">
      <timeout_messages>3600000</timeout_messages>
      <store>
        <remote_parties>
          <remote_party>
            <name>Supplier1</name>
            <id>90B3D51F30010000</id>
```

**Figure 6 – Scenario Configuration file overview.**

### 3.1.1 Console and Log Output

This is the component where the output configurations should be set. The <default_output> component is an aggregator of the execution outputs configuration. There are two configurable execution outputs: the console output and the execution log file.

The only configurable option available both for console output and execution log file is the verbose level. The higher the value chosen, the more information will be shown. This parameter may be set using the attribute "level".

```
    <default_output class="CAggregOutputter" level="200" name="OutputManager">
      <output class="CATGConsoleOutputter" level="4" name="ConsoleOutputter" />
      <output class="CATGBufferedFileOutputter" level="100" name="FileOutputter" />
    </default_output>
```

**Figure 7 – Output Configuration overview.**

The “class” and “name” element should not be changed by the Tester. The element “class” is the class that will be used in the source code of the testing tool. The element “name” is the name of the object that will be created from the specified class.

In Figure 7, the verbose level for the console is level=“4”; for the execution log the verbose level is defined as level=“100”.

Table 2 presents the available options for the verbose level as well as a description about the meaning of the value and recommended usage.

| Verbose Level | Description |
|---|---|
| 0 | Only core information will be outputted. Lowest verbose level available. |
| 2 | Both core and high criticality information will be outputted/recorded. |
| 4 | Core, high and medium criticality information will be outputted/recorded. |
| 6 | Core, high, medium and low criticality information will be outputted/recorded. |
| 100 | Core, high, medium and low criticality information will be outputted / recorded. Also non-critical information included only for debug purposes will be outputted / recorded. |
| Custom | Other values can be used, as long as are in the range [0...127]. |

**Table 2 – Available verbose level configuration values.**

### 3.1.2 Test Case Scheduling

This is the component where the scheduling configurations should be set. The Scheduler manages the order in which each Test Cases is executed. In this component the Tester should configure the attribute “class” associated with the Scheduler.

```
    <scheduler class="CSchedulerTestCaseInterleaving" name="Scheduler" />
```

**Figure 8 – Scheduling Configuration overview.**

Table 3 presents the available options for the scheduling configuration.

| Type | Class | Description |
|---|---|---|
| Test Case Interleaving | CSchedulerTestCaseInterleaving | All iterations of a Test Case will be executed sequentially before moving to the next Test Case. |
| Iteration Interleaving | CSchedulerIterationInterleaving | One iteration of each Test Case will be executed sequentially until the last Test Case is reached. Then the execution will move on to the second iteration of the first Test Case and so forth. |

**Table 3 – Available scheduling configuration values.**

### 3.1.3 Result Manager

This is the component where the result configurations should be placed. The result manager is responsible for generating the reports of the Test execution. This is the only option available so far and produces the raw (XML) and the formatted (HTML) reports described in Sections 5.3 and 5.4. Advice is made for the Tester to not change this component.

```
<result_manager class="CGenericResultManager" name="ResultManager" />
```

**Figure 9 – Result Manager configuration overview.**

### 3.1.4 Simulation Manager

The Simulation Manager component is responsible for the real-time execution coordination of all the Simulators and equipment. This is a feature of VSIS Core which is not applicable in this scope and therefore no configuration is required. Nevertheless, the configuration file entry is kept for tool coherence purposes. Advice is made for the Tester to not change this component.

```
<equipment class="CSimulationManager" name="SimulationManager">
  <period>100</period>
</equipment>
```

**Figure 10 – Simulation Manager configuration overview.**

### 3.1.5 Emulators

Emulators are the software components with coded logic that mimic the behaviour of the real metering devices, namely ESME, GSME, CHF and KRPs. Even when real metering equipment is used some emulators are required to reproduce the full system's functionality, such is the case of the KRPs and ACB.

```
<equipment class="CACBEmulator" name="ACBEmulator">
  <timeout_messages>3600000</timeout_messages>
</equipment>

<equipment class="CRemotePartyEmulator" name="RPEmulator">
  <timeout_messages>3600000</timeout_messages>
  <store>
    <remote_parties>
      <remote_party>
        <name>Supplier1</name>
        <id>90B3D51F30010000</id>
```

**Figure 11 – Remote Party Emulators configuration overview.**

The next sections detail the different types of emulators used in the framework and respective configurable parameters, however they all share the common “timeout_messages” element that can be configured to change how long the GFI tool will wait for messages. In most cases, changing the timeout for the ACB emulator will be sufficient for changing the timeout when waiting for messages from the meter to be received by the tool.

#### Access Control Broker

The ACB emulator does not require any particular configurations. Advice is made for the Tester to not change this component.

```
<equipment class="CACBEmulator" name="ACBEmulator">
  <timeout_messages>3600000</timeout_messages>
</equipment>
```

Figure 12 – ACB Emulator configuration overview.

### Remote Party

This component emulates the Known Remote Parties (KRP), for example, Supplier, ACB and Network Operator. For each KRP there are a number of configuration options available that are nested under the element <remote_party>.

```
<equipment class="CRemotePartyEmulator" name="RPEmulator">
  <timeout_messages>3600000</timeout_messages>
  <corrupted_messages>false</corrupted_messages>
  <store>
    <remote_parties>
      <remote_party>
        <name>Supplier1</name>
        <id>90B3D51F30010000</id>
        <counters>
          <originator_counter>4981</originator_counter>
        </counters>
        <private_keys>
          <digital_signature_file>
            ./conf/smki/org/ZAZ1/key/Z1-supplier-ds.key
          </digital_signature_file>
          <key_agreement_file>
            ./conf/smki/org/ZAZ1/key/Z1-supplier-ka.key
          </key_agreement_file>
          <pre_payment_top_up_file>
            ./conf/smki/org/ZAZ1/key/Z1-supplierPP-ka.key
          </pre_payment_top_up_file>
        </private_keys>
        <certificates>
          <key_agreement_certificate_file>
            ./conf/smki/org/ZAZ1/cert/Z1-supplier-ka.pem
          </key_agreement_certificate_file>
        </certificates>
      </remote_party>
      <remote_party>
        <name>NetworkOperator1</name>
```

Figure 13 – Remote Party configuration overview.

Although the selection of the Remote Party is made in the Test Properties file (one of the inputs from Table 1) the definitions for each of them is kept in the Scenario file.

Table 4 presents the configurations available for each KRP.

| Name | XML element | Description |
|---|---|---|
| Name | <name> | Device name as in GBCS documentation. The Use Case's specification uses this name to refer to the KRP to be used. |
| ID | <id> | Entity ID as in GBCS documentation. |
| Originator Counter | <originator_counter> | Initial value for the originator counter. Incremented for each message sent by the KRP. |
| UTRN Counter | <utrn_counter> | Initial value for the UTRN counter. |
| Private Keys | <digital_signature_file> | Path to the .key file containing the private and public key as provided in the SMKI pack and |

| Name | XML element | Description |
|---|---|---|
| | <key_agreement_file><br><pre_payment_top_up_file> | used by the KRP to construct and verify MAC, signature and pre-payment top up where applicable. |
| Certificates | <key_agreement_certificate_file><br><digital_signature_certificate_file> | Path to the .crt certificate file as provided in the SMKI pack. Used for Supplementary Remote Party operations when applicable. |
| Force Corrupted Messages | <corrupted_messages> | This component allows to force corruption of GBCS messages at the atg-testing tool. |

**Table 4 – Remote Party configuration values.**

This component also contains information about which Devices the referred Remote Parties are allowed to communicate with. For each Device there are a number of configurations available that are nested under the element <device>.

```
<devices>
  <device>
    <name>ESME1</name>
    <id>00DB1234567890A0</id>
    <public_keys>
      <digital_signature_certificate_file>
        ./conf/smki/device/device-ee/cert/00db1234567890A0-ds.der
      </digital_signature_certificate_file>
      <key_agreement_certificate_file>
        ./conf/smki/device/device-ee/cert/00db1234567890A0-ka.der
      </key_agreement_certificate_file>
    </public_keys>
  </device>
  <device>
    <name>GSME1</name>
    <id>00DB1234567890A1</id>
    <public_keys>
```

**Figure 14 – Remote Party Devices configuration overview.**

Table 5 presents the configurations available for each Device the Remote Party is allowed to communicate with.

| Name | XML element | Description |
|---|---|---|
| Name | <name> | Device name as in GBCS documentation. The Use Case's specification uses this name to refer to the Device to be used. |
| ID | <id> | Entity ID as in GBCS documentation. |
| Public Keys | <digital_signature_certificate_file><br><key_agreement_certificate_file> | Path to the .der file containing the device certificate, as provided in the SMKI pack. Used to verify signatures and MAC, also to decrypt encrypted messages. |

**Table 5 – Remote Party configuration values.**

### ESME, GSME,CHF, GPF, PPMID, HCALCS

These components are used only in emulated scenarios. When using real physical devices these components will not be present in the Scenario Configuration file. For each Device there are a number of configurations available that are nested under the element <device>. Importantly, it's used to define the state of the trust anchor cells, device certificates and the value of select SMETS attributes.

```
<equipment class="CESMEEmulator" name="ESMEEmulator">
  <timeout_messages>3600000</timeout_messages>
  <store>
    <devices>
      <device>
        <id>00DB1234567890A0</id>
        <counters>
          <originator_counter>2000</originator_counter>
        </counters>
        <private_keys>
          <digital_signature_file>
            ./conf/smki/device/device-ee/key/00db1234567890A0-ds.key
          </digital_signature_file>
          <key_agreement_file>
            ./conf/smki/device/device-ee/key/00db1234567890A0-ka.key
          </key_agreement_file>
        </private_keys>
        <certificates>
```

**Figure 15 – Device emulator configuration overview.**

Table 6 presents the configurations available for each Device emulator.

| Name | XML element | Description |
|---|---|---|
| ID | <id> | Entity ID as in GBCS documentation. |
| Originator Counter | <originator_counter> | Initial value for the originator counter. Incremented for each message sent by the Device. |
| Private Keys | <digital_signature_file><br><key_agreement_file> | Path to the .key file containing the private and public key, as provided in the SMKI pack and used by the Device to construct and verify MAC, signature and encrypt sensitive data. |
| Certificates | <digital_signature_certificate_file><br><key_agreement_certificate_file> | Path to the .der file containing the device certificate, as provided in the SMKI pack. Used to be provided to a KRP upon request. |
| Certification Request | <digital_signature_certificate_file><br><key_agreement_certificate_file> | Path to the .csr file, as provided in the SMKI pack . Used to be provided to a KRP upon request for certification issuing. |
| Trust Anchor Cell Name | <name> | KRP name, the certificate name and the role separated by commas. |

| Name | XML element | Description |
|---|---|---|
| Trust Anchor Cell Remote Party Role | <remote_party_role> | The class value of Remote Party in relation to the Device is capable of storing Security Credentials, as specified in GBCS. |
| Trust Anchor Cell Key Usage | <key_usage> | The key usage value of Remote Party in relation to the Device is capable of storing Security Credentials, as specified in GBCS. |
| Trust Anchor Cell Cell Usage | <cell_usage> | The cell usage value of Remote Party in relation to the Device is capable of storing Security Credentials, as specified in GBCS. |
| Trust Anchor Cell Certificate | <certificate_file> | Path to the .crt file containing the KRP certificate, as provided in the SMKI pack. Used by the Device to verify MAC, signature and decrypt sensitive data. |
| SMETS Object Storage | <smets> | Data regarding the SMETS objects used by the emulator in an emulated scenario. |

**Table 6 – Meter Device emulator configuration values.**

Some SMETS Objects require Storage for purposes of Test Case sequencing where emulated metering devices are used. These objects will be kept nested under the element <smets> as shown in Figure 16.

```
<smets>
  <DeviceLogEntries>[
                    {
                    "DeviceEntityIdentifier": "01 02 03 04 05 06 07 08",
                    "DeviceType": 3
                    },
                    {
                    "DeviceEntityIdentifier": "00 DB 12 34 56 78 90 99",
                    "DeviceType": 3
                    },
                    {
                    "DeviceEntityIdentifier": "00 DB 12 34 56 78 90 97",
                    "DeviceType": 5
                    }
                    ]</DeviceLogEntries>
  <MeterBalance>100</MeterBalance>
  <MeterBalance_value>250000</MeterBalance_value>
  <PaymentMode_value_PaymentMode>2</PaymentMode_value_PaymentMode>
  <RMSVoltageSagThreshold_threshold>216</RMSVoltageSagThreshold_threshold>
```

**Figure 16 – Meter Device emulator store model overview.**

### CHF

This component emulates the Communications Hub Function and should only be present when running an emulated scenario. The information configurable in this component follows the same tract as the ESME and GSME emulator components. That is, defines the security certificates and keys, and the relevant CHTS attributes.

```xml
<equipment class="CCHFEmulator" name="CHFEmulator">
  <timeout_messages>3600000</timeout_messages>
  <store>
    <devices>
      <device>
        <id>00DB1234567890A2</id>
        <counters>
          <originator_counter>4000</originator_counter>
        </counters>
        <private_keys>
          <digital_signature_file>
            ./conf/smki/device/device-ee/key/00db1234567890A2-ds.key
          </digital_signature_file>
          <key_agreement_file>
            ./conf/smki/device/device-ee/key/00db1234567890A2-ka.key
          </key_agreement_file>
        </private_keys>
```

**Figure 17 – CHF Emulator configuration overview.**

Table 7 describes the configurable options for the emulated CHF component.

| Name | XML element | Description |
|---|---|---|
| ID | <id> | Entity ID as in GBCS documentation. |
| Originator Counter | <originator_counter> | Initial value for the originator counter. Incremented for each message sent by the Device. |
| Private Keys | <digital_signature_file><br><key_agreement_file> | Path to the .key file containing the private and public key, as provided in the SMKI pack and used by the Device to construct and verify MAC, signature and encrypt sensitive data. |
| Certificates | <digital_signature_certificate_file><br><key_agreement_certificate_file> | Path to the .der file containing the device certificate, as provided in the SMKI pack. Used to be provided to a KRP upon request. |
| Certification Request | <digital_signature_certificate_file><br><key_agreement_certificate_file> | Path to the .csr file, as provided in the SMKI pack . Used to be provided to a KRP upon request for certification issuing. |
| Trust Anchor Cell Name | <name> | KRP name, the certificate name and the role separated by commas. |
| Trust Anchor Cell Remote Party Role | <remote_party_role> | The class value of Remote Party in relation to the Device is capable of storing Security Credentials, as specified in GBCS. |
| Trust Anchor Cell Key Usage | <key_usage> | The key usage value of Remote Party in relation to the Device is capable of storing Security Credentials, as specified in GBCS. |

| Name | XML element | Description |
|---|---|---|
| Trust Anchor Cell Cell Usage | <cell_usage> | The cell usage value of Remote Party in relation to the Device is capable of storing Security Credentials, as specified in GBCS. |
| Trust Anchor Cell Certificate | <certificate_file> | Path to the .crt file containing the KRP certificate, as provided in the SMKI pack. Used by the Device to verify MAC, signature and decrypt sensitive data. |
| CHTS Object Storage | <chts> | Data regarding the CHTS objects used by the emulator in an emulated scenario. |

**Table 7 – CHF emulator configuration values.**

The CHTS attributes that are supported by the CHF emulator are as in Figure 18, and described in Table 8.

```
<chts>
  <DeviceIdentifierCHF>xyz</DeviceIdentifierCHF>
  <FirmwareVersion_value>1234</FirmwareVersion_value>
  <DeviceLogCHF_hanIdentifier>
    00 01 02 03 04 05 06 07
  </DeviceLogCHF_hanIdentifier>
  <DeviceLogCHF_entries>
    <entry>
      <entityIdentifier>
        00 DB 12 34 56 78 90 A0
      </entityIdentifier>
      <installationCredentials>
        83 FE D3 40 7A 93 97 23 A5 C6 39 B2 69 16 D5 06
      </installationCredentials>
      <deviceType>0</deviceType>
    </entry>
    <entry>
      <entityIdentifier>00 DB 12 34 56 78 90 A1</entityIdentifier>
      <installationCredentials>
        83 FE D3 40 7A 93 97 23 A5 C6 39 B2 69 16 D5 05
      </installationCredentials>
      <deviceType>1</deviceType>
    </entry>
  </DeviceLogCHF_entries>
</chts>
```

**Figure 18 – CHTS Attributes.**

| Name | XML element | Description |
|---|---|---|
| CHTS Object Storage | <chts> | Data regarding the SMETS objects used by the CHF. |
| CHTS Entries<br>Entity Identifier | <entityIdentifier> | Entity Identifier of the Device in the Whitelist of the CHF. |
| CHTS Entries | <installationCredentials> | Installation credentials (installation code) of the Device in the Whitelist of the CHF. |

| Name | XML element | Description |
|---|---|---|
| Installation Credentials | | |
| CHTS Entries<br>Device Type | <deviceType> | Type of device in the Whitelist of the CHF. |

Table 8 – CHTS attributes configuration values.

### GPF

This component emulates the Gas Proxy Function and should only be present when running an emulated scenario where the GSME emulated component is also present. The information configurable in this component follows the same tract as the ESME and GSME emulator components. That is, defines the security certificates and keys. However, as the GPF will mirror the SMETS data on the GSME, many of the SMETS attributes do not need to be defined in this component.

```
<equipment class="CGPFEmulator" name="GPFEmulator">
  <timeout_messages>3600000</timeout_messages>
  <store>
    <mirrorid>00DB1234567890A1</mirrorid>
    <devices>
      <device>
        <id>00DB1234567890A3</id>
        <counters>
          <originator_counter>3000</originator_counter>
        </counters>
        <private_keys>
          <digital_signature_file>
            ./conf/smki/device/device-ee/key/00db1234567890A3-ds.key
          </digital_signature_file>
          <key_agreement_file>
            ./conf/smki/device/device-ee/key/00db1234567890A3-ka.key
          </key_agreement_file>
        </private_keys>
```

Figure 19 – GPF Emulator configuration overview.

In addition to Table 6, Table 9 presents the new configuration options for the GPF.

| Name | XML element | Description |
|---|---|---|
| Mirror ID | <mirrorid> | Entity ID of the GSME being mirrored. |

Table 9 – GPF emulator configuration values.

The only supported SMETS attributes for the GPF emulator is the device log. This is represented as depicted in Figure 20.

```
<smets>
  <DeviceLogEntries>[
                    {
                    "DeviceEntityIdentifier": "01 02 03 04 05 06 07 08",
                    "DeviceType": 6
                    },
                    {
                    "DeviceEntityIdentifier": "00 DB 12 34 56 78 90 99",
                    "DeviceType": 5
                    },
                    {
                    "DeviceEntityIdentifier": "00 DB 12 34 56 78 90 97",
                    "DeviceType": 5
                    }
                    ]
  </DeviceLogEntries>
</smets>
```

**Figure 20 – GPF Emulator device log.**

### 3.1.6 Client

The CHF client is used in-situ of the respective emulator when running a test against physical meters using the gfi-comms-hub. Since previous versions of GFI, the CHF section no longer contains all the ZigBee related configuration as it is now part of the Communications Hub.

#### CHF

This component configures the Communications Hub client and should only be present when running a scenario with physical devices. The information configurable in this component is slightly different as the actual configuration of the CHF is part of the communication hub daemon.

```
<equipment class="CCHFClient" name="CHFClient">
  <chf_port>50001</chf_port>
  <store>
    <devices>
      <device>
        <id>00DB1234567890A2</id>
        <counters>
          <originator_counter>4000</originator_counter>
        </counters>
        <private_keys>
          <digital_signature_file>
            ./conf/smki/device/device-ee/key/00db1234567890A2-ds.key
          </digital_signature_file>
          <key_agreement_file>
            ./conf/smki/device/device-ee/key/00db1234567890A2-ka.key
          </key_agreement_file>
        </private_keys>
        <certificates>
          <digital_signature_certificate_file>
            ./conf/smki/device/device-ee/cert/00db1234567890A2-ds.der
          </digital_signature_certificate_file>
          <key_agreement_certificate_file>
            ./conf/smki/device/device-ee/cert/00db1234567890A2-ka.der
          </key_agreement_certificate_file>
        </certificates>
```

**Figure 21 – CHF Client configuration overview.**

Table 10 presents the new configuration options for the CHF. By design, all cryptography and message protections are handled by the GFI testing tool, thus the CHClient is required to be configured with CHF credentials and the relevant trust anchor cells.

| Name | XML element | Description |
|---|---|---|
| CHF Control Port | <chf_port> | Used by GFI for controlling the Communications Hub, the port should align with the CH configuration file. The default value for this is 50001, and should only be changed if there is a need. |
| ID | <id> | Should be the same as the CHF ID in the CH configuration file. This attribute should be different from the CH IEEE radio address. |

**Table 10 – CHF emulator configuration values.**

## 3.2 Test Properties File

This Section describes the Test Properties file referred to as <properties-file>.xml. This file contains all the Test Cases that constitute the Test as well as information regarding the Test procedure, like the Test name, identifier and purpose. Any other information can be added to the Test Properties file as explained in the following subsections, for instance the Manufacturer's Device name, the Device firmware version, etc. This information will be added to the Test Report as is. In the following subsections a detailed description of all the components in this file is presented.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE executionconfiguration []>
<executionconfiguration>
    <testmanager master_node="svfmaster">
        <testspec class="CTestSpec" name="ATG-TEST-ECS40">
            <report>
                <item name="USE CASE">ECS40</item>
                <item name="TEST ID">ATG-TEST-ECS40</item>
                <item name="TEST PURPOSE">Test Use Case ECS40 Read MPAN Value on the
ESME&quot;, which is for reading the MPAN on the ESME.</item>
            </report>

            <testcase class="CTestCaseECS40" name="ATG-TC-ECS40">
                <report>
                    <item name="TEST CASE ID">ATG-TC-ECS40</item>
                    <item name="TEST CASE PURPOSE">The purpose of this test case is to
validate Use Case ECS40 &quot;Read MPAN Value on the ESME&quot; by validating the
construction of the command used for reading the power registers on the ESME and its
respective response.</item>
                    <item name="TEST CASE ASSUMPTIONS AND CONSTRAINTS">GIST Issue #3145:
&quot;ECS40 - Notes for attribute MeterPointAdministrationNumber(combinedMPANS)&quot;</item>
                    <item name="AUTHOR">CSW</item>
                    <item name="VERSION">01</item>
                </report>

                <resource_combiner class="CLinearCombinations" name="linearCombinator"/>

                <resourcepool class="CResourcePoolString" name="BusinessOriginatorName">
                    <value>ACB</value>
                    <value>Supplier1</value>
                </resourcepool>

                <resourcepool class="CResourcePoolString" name="BusinessTargetName">
                    <value>ESME1</value>
                    <value>ESME1</value>
                </resourcepool>
```

**Figure 22 – Test Properties file overview.**

### 3.2.1 Report Information

This component, identified by <report> in Figure 22, holds the information to be printed in the Test Report. The names of the fields are rather self-explanatory (Use Case, Test ID, test purpose) regarding the information they should store. The Tester is free to include any other information that is required to be added to the Test Report. This information is printed as is, without any type of validation.

The same applies for each Test Case. The component <report> is also supported and any type of information can be added to it. Figure 23 presents an example of such information, namely Test Case ID, Purpose, Assumptions and Constraints, Author and Version as shown in Figure 23 below.

```
        <testspec class="CTestSpec" name="ATG-TEST-ECS40">
            <report>
                <item name="USE CASE">ECS40</item>
                <item name="TEST ID">ATG-TEST-ECS40</item>
                <item name="TEST PURPOSE">Test Use Case ECS40 Read MPAN Value on the
ESME&quot;, which is for reading the MPAN on the ESME.</item>
            </report>

            <testcase class="CTestCaseECS40" name="ATG-TC-ECS40">
                <report>
                    <item name="TEST CASE ID">ATG-TC-ECS40</item>
                    <item name="TEST CASE PURPOSE">The purpose of this test case is to
validate Use Case ECS40 &quot;Read MPAN Value on the ESME&quot; by validating the
construction of the command used for reading the power registers on the ESME and its
respective response.</item>
                    <item name="TEST CASE ASSUMPTIONS AND CONSTRAINTS">GIST Issue #3145:
&quot;ECS40 - Notes for attribute MeterPointAdministrationNumber(combinedMPANS)&quot;</item>
                    <item name="AUTHOR">CSW</item>
                    <item name="VERSION">01</item>
                </report>
```

**Figure 23 – Test Properties Report Information overview.**

### 3.2.2 Resource Combiner

The Resource Combiner, identified by <resource_combiner> in Figure 24, allows the generation of Test Case iterations based on the combinations of the elements in each resource pool. In the scope of ATG/GFI no automatic combinations are used. Instead, a Linear Combiner is used, meaning that no combination is made using the resource pools of the Test Case. Any required resource combination should be added manually by inserting a new value in each resource pool entry. Advice is made for the Tester to not change this component.

```
                <resource_combiner class="CLinearCombinations" name="linearCombinator"/>
```

**Figure 24 – Test Properties Resource Combiner overview.**

### 3.2.3 Resource Pool

These are the parameters to be used in the Test Case, identified by <resourcepool> in Figure 25. They are used as Inputs to build the Command messages or Expected Results to perform checks in the Response message fields. Using the Linear Combiner, each iteration will use the respective data element of each resource pool in an orderly fashion. Thus, every resource pool should have the same number of data elements.

```
                <resourcepool class="CResourcePoolString" name="BusinessOriginatorName">
                    <value>ACB</value>
                    <value>Supplier1</value>
                </resourcepool>

                <resourcepool class="CResourcePoolString" name="BusinessTargetName">
                    <value>ESME1</value>
                    <value>ESME1</value>
                </resourcepool>
```

**Figure 25 – Test Properties Resource Pool overview.**

### 3.2.4 Test Repetition

This component, identified by <run> in Figure 26, is used by the framework to execute a Test a specified number of times using the attribute “times” (e.g. for stress testing or performance purposes).

```
                <resourcepool class="CResourcePoolString"
                              name="MeterAdministrationPointNumber(exportMPAN).result">
                    <value>0</value>
                    <value>0</value>
                </resourcepool>

            </testcase>
        </testspec>

        <run times="1" />
    </testmanager>
</executionconfiguration>
```

**Figure 26 – Test Repetition Counter overview.**

# 4 Communications Hub

The Communications Hub is a separate piece of software from the main testing tool, and intended to run as a daemon, which instances of the GFI testing tool will connect to and request it to proxy messages back and forth to the end device on the HAN.

This section describes the configuration of the Communications Hub and how to interact with the daemon.

## 4.1 Configuration File

The Communications Hub has a singular configuration file. This file is responsible for defining the ZigBee network and any pre-configured HAN whitelist (the whitelist can be updated using CCS0x usecases, please refer to GBCS for more information).

The best way to get a sample configuration file is to use the GUI to configure a Communications Hub and use the export function to save the configuration to a XML file as show in Figure 27 – Export Communications Hub configuration file.

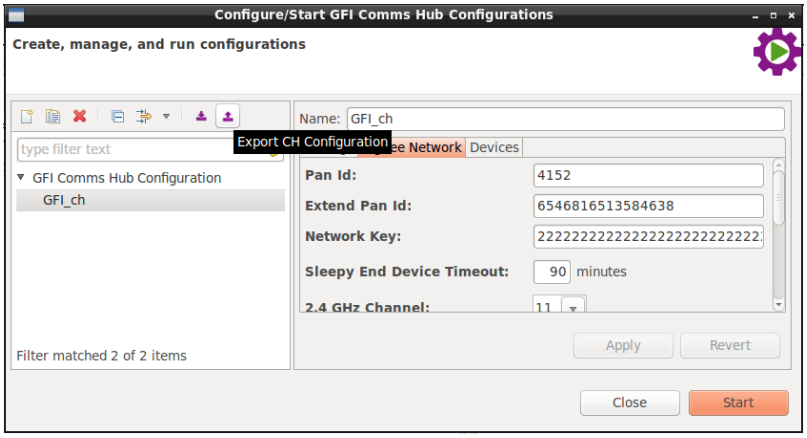


Figure 27 – Export Communications Hub configuration file.

**Proviso:**

When configuring the CH, please ensure the following:

- CHF EUI64 is virtual, the user is recommended to select a unique ID from the DCC SMKI provided with GFI (located at `/opt/gfi/conf/smki/device/`).
- GPF EUI64 is the CH IEEE radio address, as displayed on the GFI kit yellow label.

### 4.1.1 Interfaces Configuration

```
<CommsHubConfiguration>
  <Port>/dev/ttyUSB0</Port>
  <BaudRate>57600</BaudRate>
  <FlowControl>rtscts</FlowControl>
  <ControlPort>50000</ControlPort>
  <CHFPort>50001</CHFPort>
  <SCPICHFPort>50002</SCPICHFPort>
  <FirmwareVersion>GFIv2.0</FirmwareVersion>
  <TimeoutMessages>3600000</TimeoutMessages>
```

Figure 28 – Interfaces configuration overview.

The configurable options are described in Table 11.

| Name | XML element | Description |
|---|---|---|
| CHF Control Port | ControlPort | TCP port used for controlling the daemon. For example, the GUI uses it to get a list of which devices are on the HAN. Suggested to use 50000. |
| CHF data Port | CHFPort | TCP port to which the gfi-testing-tool CHFClient connects. Used for transferring GBCS messages. Suggested to use 50001. |
| SCPI CHF | SCPICHFPort | TCP port to which an external tool can connect as a CHFClient and use SCPI syntax for TCP message exchange. Used for transferring GBCS messages. Suggested to use 50002. |
| Serial Port | Port | The serial port where the USB ZigBee adapter is connected. |
| Serial port speed | BaudRate | Speed of the serial port where the USB ZigBee adapter is connected. Depends on the apater firmware configuration. Default is 57600. |
| Serial port Flow Control | FlowControl | The type of flow control to use in the serial port. |

Table 11 – Interfacesconfiguration values.

### 4.1.2 HAN Configuration

```
<ZigbeeNetwork>
  <PanId>1234</PanId>
  <ExtendedPanId>0011223344556677</ExtendedPanId>
  <NetworkKey>22222222222222222222222222222222</NetworkKey>
  <PermitJoin>1</PermitJoin>
  <EndDeviceTimeout>5</EndDeviceTimeout>
  <JoinTimeout>255</JoinTimeout>
  <GHzChannel>11</GHzChannel>
  <SubGHzChannel>2</SubGHzChannel>
  <SubGHzBand>863-876MHz</SubGHzBand>
  <GHzRadioTxPower>3</GHzRadioTxPower>
  <SubGHzRadioTxPower>3</SubGHzRadioTxPower>
  <Mode>Dual</Mode>
</ZigbeeNetwork>
```

Figure 29 – CH Configuration overview.

The configurable options are described in Table 12.

| Name | XML element | Description |
|---|---|---|
| ZigBee Network Pan ID | PanID | The ZigBee network Pan ID |
| ZigBee Network Extended Pan ID | ExtendedPanID | The ZigBee network extended Pan ID |
| ZigBee Network Network Key | NetworkKey | The ZigBee network key |
| ZigBee Network Permit Join | PermitJoin | Allows the join of devices to the HAN (or not). |
| ZigBee Network Join Timeout | JoinTimeout | The join timeout, specified in seconds. |
| ZigBee End-Device timeout | EndDeviceTimeout | Timeout for end-devices after which the parent removes it from the end childs list and devide needs to rejoin network. For compatibility with end devices using NXP stacks, it is recommended to set it to 90 min. |
| ZigBee Network Channel for 2.4GHz | GHzChannel | The ZigBee network channel for 2.4GHz. Suggested to use a ZSE preferred channel when possible. |
| ZigBee Network Tx power for 2.4GHz | GHzRadioTxPower | The power of radio for transmitting data over the 2.4GHz |
| ZigBee Network band for subGHz | SubGHzBand | The band of subGHz to use. Low band (838-876MHz) or high band (915-921MHz) |
| ZigBee Network Channel for subGHz | SubGHzChannel | The ZigBee network channel for subGHz. Low band (0-48), high band (0-12) |
| ZigBee Network Tx power for subGHz | SubGHzRadioTxPower | The power of radio for transmitting data over the subGHz |
| ZigBee Network Mode | Mode | The ZigBee network mode to form (GHz, SubGHz or Dual). Default: Dual |

**Table 12 – CH HAN configuration values.**

### 4.1.3 GPF configuration

The GPF configuration consists in the GFI ID, its private key and remote parties certificates.See Figure 30 for an example. Several TrustAnchorCell elements can be included under TrustAnchorCells

```
<GPF>
  <Id>00DB1234567890A3</Id>
  <PrivateKeys>
    <KeyAgreementFile>./conf/smki/device/device-ee/key/00db1234567890A3-ka.key</KeyAgreementFile>
    <DigitalSignatureFile>./conf/smki/device/device-ee/key/00db1234567890A3-ds.key</DigitalSignatureFile>
  </PrivateKeys>
  <TrustAnchorCells>
    <TrustAnchorCell>
```

```
      <Name>Root, Key Cert, Management</Name>
      <RemotePartyRole>0</RemotePartyRole>
      <KeyUsage>32</KeyUsage>
      <CellUsage>0</CellUsage>
      <CertificateFile>./conf/smki/org/ZAZ1/cert/OCARoot-ZA.pem</CertificateFile>
    </TrustAnchorCell>
  </TrustAnchorCells>
</GPF>
```

**Figure 30 – GPF Configuration overview.**

The configurable options are described in Table 13.

| Name | XML element | Description |
|---|---|---|
| GPF EUI64 | Id | Entity ID as in GBCS documentation, see proviso above. |
| GPF Key Agreement | KeyAgreementFile | The path relative to the running directory for the private key agreement file to |
| GPF Digital Signature | DigitalSignatureFile | The path relative to the running directory for the digital signature file to |
| GPF TrustAnchorCell name | Name | Identification of a Trust Anchor Cell {remotePartyRole, keyUsage, cellUsage} See GBCS |
| GPF TrustAnchorCell remote party role | RemotePartyRole | Identifyes the  Remote Party Role this Cell relate to. See GBCS |
| GPF TrustAnchorCell Key Usage | KeyUsage | Identifyes the cryptographic use of the Public Key in this Cell. See GBCS |
| GPF TrustAnchorCell Cell Usage | CellUsage | Identifyes the Public Keys of the same keyUsage for the same Remote Party Role. See GBCS |
| GPF TrustAnchorCell Certificate | CertificateFile | The path relative to the running directory for the certificate file |

**Table 13 – GPF configuration values.**

### 4.1.4 CHF configuration

The CHF configuration consists in the CHF ID, its private key and remote parties certificates.See Figure 31 for an example. Several TrustAnchorCell elements can be included under TrustAnchorCells

```
<CHF>
  <Id>00DB1234567890A2</Id>
  <OtaStorage>/home/Desktop/ota-files</OtaStorage>
  <PrivateKeys>
    <KeyAgreementFile>./conf/smki/device/device-ee/key/00db1234567890A2-ka.key</KeyAgreementFile>
    <DigitalSignatureFile>./conf/smki/device/device-ee/key/00db1234567890A2-ds.key</DigitalSignatureFile>
  </PrivateKeys>
  <TrustAnchorCells>
    <TrustAnchorCell>
      <Name>Root, Key Cert, Management</Name>
      <RemotePartyRole>0</RemotePartyRole>
```

```
      <KeyUsage>32</KeyUsage>
      <CellUsage>0</CellUsage>
      <CertificateFile>./conf/smki/org/ZAZ1/cert/OCARoot-ZA.pem</CertificateFile>
    </TrustAnchorCell>
  </TrustAnchorCells>
</CHF>
```

**Figure 31 – CHF Configuration overview.**

The configurable options are described in Table 14.

| Name | XML element | Description |
|---|---|---|
| CHF EUI64 | Id | Entity ID as in GBCS documentation, see proviso above. |
| CHF OTA Storage | OtaStorage | The absolute path to the folder that stores the OTA images to send to the devices.<br>This field is optional and the default path is: /opt/gfi/ota-files |
| CHF Key Agreement | KeyAgreementFile | The path relative to the running directory for the private key agreement file to |
| CHF Digital Signature | DigitalSignatureFile | The path relative to the running directory for the digital signature file to |
| CHF TrustAnchorCell name | Name | Identification of a Trust Anchor Cell {remotePartyRole, keyUsage, cellUsage} See GBCS |
| CHF TrustAnchorCell remote party role | RemotePartyRole | Identifyes the Remote Party Role this Cell relate to. See GBCS |
| CHF TrustAnchorCell Key Usage | KeyUsage | Identifyes the cryptographic use of the Public Key in this Cell. See GBCS |
| CHF TrustAnchorCell Cell Usage | CellUsage | Identifyes the Public Keys of the same keyUsage for the same Remote Party Role. See GBCS |
| CHF TrustAnchorCell Certificate | CertificateFile | The path relative to the running directory for the certificate file |

**Table 14 – CHF configuration values.**

### 4.1.5 CHF Device Log

It is possible to pre-configure a device log in the configuration file for the Communications Hub, this is achieved by adding the “DeviceLogCHFEntries” element to the configuration file, which can contain a number of device log entries. See Figure 50 for an example, and Table 15 for an explanation.

```
  <DeviceLogCHFEntries>
    <Entry>
      <EntityIdentifier>00DB1234567890A1</EntityIdentifier>
      <InstallationCredentials>83FED3407A939723A5C639B200000000</InstallationCredentials>
      <DeviceType>0</DeviceType>
    </Entry>
  </DeviceLogCHFEntries>
</CommsHubConfiguration>
```

**Figure 32 – CH Device Log configuration overview.**

| Name | XML element | Description |
|---|---|---|
| Entity Identifier | <entityIdentifier> | Entity Identifier of the Device in the Whitelist of the CHF. |

| Name | XML element | Description |
|---|---|---|
| Installation Credentials | <installationCredentials> | Installation credentials (installation code) of the Device in the Whitelist of the CHF. The following are supported: 8-byte install, 10-byte install (8+CRC), 16-byte install, and 18-byte install (16 + CRC). |
| Device Type | <deviceType> | Type of device in the Whitelist of the CHF. The supported values are: 0 for ESME, 1 for GSME, 13 for ALCS, 128 for IHD, 130 for CAD, 132 for PPMID. |

**Table 15 – CH Device Log attributes configuration values.**

## 4.2 Execution

Before starting the communications hub, please ensure that the ZigBee radio is attached to the computer, and that the configuration file has been created. The simplest execution of the tool is shown in Figure 33, where the user has first navigated to "`/opt/gfi/`"

```
~$ cd /opt/gfi
/opt/gfi$ ./gfi-comms-hub ~/Desktop/my-ch-configuration.xml
-----------------------------------------------------------
STARTING EXECUTION TIME: Wed Feb 21 09:27:35 2018
-----------------------------------------------------------
```

**Figure 33 – GFI communications hub command line invocation example.**

The usage of the tool is as follows, where the supported options are described in Table 16:

```
gfi-comms-hub <configuration-file.xml> [options]
```

| Option | Description |
|---|---|
| `configuration-file.xml` | The configuration file, as specified by Section 4.1. |
| `--loglevel=n` | Sets the console log level to level n, defaults to 4 if not specified. |
| `--logdir=dir` | Sets the directory where the logfiles will be saved, defaults to current directory if not specified. If existing log files are located in the same directory, they will be removed when the tool runs. |
| `--version` | Display the tool version |
| `--hard-start` | Sets the hard start mode. The CH will not have any previous run information, so it will not restore the HAN. |

| Option | Description |
|---|---|
| `--soft-start` | Sets the soft start mode. The CH will have the previous run information and will restore the HAN. |

**Table 16 – CH command line options.**

The tool will launch a command line interface through TCP and the user can access it using the command 'telnet ip port' as shown in Figure 34.

The default port is 49999 but can be changed by modifying the "TelnetPort" field on the comms-hub configuration file as shown in Figure 32.

```
gfi@gfi:~$ telnet 127.0.0.1 49999
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
===================== Command Line Interface =====================

comms-hub>
```

**Figure 34 – GFI communications hub command line.**

The option '?' shows the available commands as shown in figure 35.

```
===================== Command Line Interface =====================

comms-hub>
  help                    Show available commands
  quit                    Disconnect
  logout                  Disconnect
  exit                    Exit from current mode
  history                 Show a list of previously run commands
  enable                  Turn on privileged commands
  shutdown                Shutdown program
  tunnel                  ...
  send                    ...
  read                    ...
  print                   ...
  tHreads                 ...
  functional-flag         ...
  keys                    ...
  channel                 ...
  lock-api                Lock API for <second> seconds
  generate-chf-event-log  Generate a random CHF EventLog at current time
  start-monitoring        Start radio signal monitoring (RSSS and LQO) with node <node>
  stop-monitoring         Stop radio signal monitoring
  set-tx-dbm-power        Set Tx dbm power. phy = 0:2.4GHz    1:SubGHz <phy> <db>
  ota                     ...
  switch-cli              Switch to Silabs CLI

comms-hub>
```

**Figure 35 - GFI communications hub command line menu.**

Some of the available commands have sub-commands which can be used by typing the command and then '?'. All the following options and/or arguments for the command will be displayed as shown in Figure 36.

```
comms-hub> tunnel
  start                 Start Tunnel <node>
  stop                  Stop Tunnel <node>
  send-message          Send message through Tunnel
                        Arguments: <force> <node> <size> <bytes>
                        Argument bytes is optional, if not set it will send
                        pseudo-random data
                        <force> Force to send queue messages [y=yes, ENTER=no]
  empty-messages        Empty tunnel messages <node>

comms-hub> tunnel
```

**Figure 36 - Command options and arguments**

The TAB key will autocomplete the commands whenever possible or show all the following options available if pressed twice after the main command is complete.

Its also possible to access the stack CLI using the command ‘switch-cli’. The command ‘custom switch-cli’ switches to the default CLI.

Please note that this menu is intended for issues debugging only since some commands may interfere with the expected comms hub logic and therefore we strongly discourage to use it in a regular situation.

Most of the commands available through this menu are also available using GBCS usecases.

## 4.3 Logging

When running the Communications Hub from a terminal window, the application will print to stdout log information, up to and including the level specified by the command line parameter “`--loglevel`”. However, all levels will be saved to the log file, that will be located in the directory specified by the “`--logdir`” command line parameter.

The format of the terminal logging, and the log files are the same as those used by the GFI testing tool, with the exception that instead of using a timestamp relative to when the application started it uses the system time. See Figure 37 for an example of this.

```
-----------------------------------------------------------------------------------------------
 STARTING EXECUTION TIME: Tue Feb 20 19:42:15 2018
-----------------------------------------------------------------------------------------------
    TIME      |  AREA  | LV | MESSAGE
-----------------------------------------------------------------------------------------------
19:42:15.817 |      CH |  0 | 2.0.0~201802061923 (Feb  6 2018 19:24:30)
19:42:15.817 |  CHSTR |  4 | Loading Comms Hub configuration from scenario/ch.xml
19:42:15.817 |  CHSTR |  4 | Configuring CH from Comms Hub Scenario data
19:42:15.817 |  CHSTR |  4 | No GPF Id in configuration. It will be use the MAC of the connected radio device
19:42:15.818 |   CBAK |  0 | Callback wrapper initialized for device COOR
19:42:15.818 |   ZBEE |  4 | Host stack version: 6.0.1.0
19:42:15.818 | COOSTR |  4 | Loading COORDINATOR configuration from Comms Hub Scenario data.
19:42:15.818 |  CBAK2 |  0 | Device specific Callback wrapper initialized for device COOR
19:42:15.818 |   CTRL |  0 | [tcp:50000] Waiting for a client
19:42:15.818 | COOSTR |  4 | GHzChannel: 11
19:42:15.818 | COOSTR |  4 | GHz RadioTxPower: 3dBm
19:42:15.819 | COOSTR |  4 | SubGHz RadioTxPower: 3dBm
19:42:15.819 | COOSTR |  4 | SubGHzChannel: 2
19:42:15.819 | COOSTR |  4 | SubGHzBand: 863-876MHz
19:42:15.819 | COOSTR |  4 | Load Device Entries...
19:42:15.819 | COOSTR |  4 | No link key, or invalid
19:42:15.819 | COOSTR |  4 | No link key, or invalid
19:42:15.819 |   ZBEE |  4 | Time zone set to Europe/London
19:42:15.820 |   ZBEE |  4 | Serial:  /dev/ttyUSB0 @ 57600 (rts/cts)
19:42:15.820 |   ZBEE |  4 | Setting Trace flags
19:42:15.820 |  STACK |  4 | Reset info: 11 (SOFTWARE)
19:42:16.988 |  STACK |  4 | ezsp ver 0x06 stack type 0x02 stack ver. [6.0.1 GA build 277]
19:42:17.014 |  STACK |  4 | Ezsp Config: set source route table size to 0x0007:Success: set
```

**Figure 37 – GFI communications hub command line output.**

## 4.4 Control Port

It is possible to interact with the Communications Hub over the control port. This mechanism is used by the GUI to request state information, and thus should not be attempted when the GUI is running, and is documented to support integration of the tool into end-user test environments.

The control port works by using JSON over TCP. The port used is that identified by the ControlPort element in the configuration file, normally port 50000 is used.

Some of the commands respond with JSON, others perform actions on the device. Currently 5 commands are supported, see Table 17 for an overview.

| Name | Parameters | Output |
|---|---|---|
| exit | none | Success/Failure |
| getKeyTable | none | Yes, JSON object representing the Key Table. |
| addDevice | eui<br>installationCredentials<br>deviceType | Success/Failure |
| removeDevice | Eui | Success/Failure |
| unJoinDevice | Eui | Success/Failure |
| getStackVersion | none | Yes, JSON with stack version from NCP and Host |
| getCHTime | none | Yes, JSON with current time in Communications Hub |
| syncTimeWithOS | none | Success/Failure |
| setCHTime | Date<br>Time | Success/Failure |
| soft-stop | none | Success/Failure |
| hard-stop | none | Success/Failure |
| getZigbeeInfo | none | Yes, JSON object with the requested values: subGhzChannel, subGhzPage, ghzChannel, nwk, panId and extendedPanId. |
| getDeviceInfo | none | Yes, JSON object with the requested values: chfId and gpfId. |

**Table 17 – CH control port actions.**

The following sections provide example executions of these commands. In all the following examples, the program `nc` is used only for demonstration purposes. The purpose of the program opens a TCP stream to the Communications Hub. It would also be possible to run the control commands from another computer connected to the network.

### 4.4.1 Exit

Requests that the application exits.

```
gfi@gfi:/opt/gfi$ echo {\"command\":\"exit\"} | nc localhost 50000
{"action": "exit"}
gfi@gfi:/opt/gfi$
```

**Figure 38 – Control Port – Exit command.**

### 4.4.2 Get Key Table

Requests the key table currently in use in the Communications Hub and the device association status. In the picture below, `json_pp` is only used to pretty print the JSON result for presentation purposes and is not required.

```
gfi@gfi:/opt/gfi$ echo {\"command\":\"getKeyTable\"} | nc localhost 50000 | json_pp
{
   "chfLogEntries" : [
      {
         "nodeId" : 65535,
         "authenticated" : false,
         "eui" : "00DB1234567890A2",
         "linkKey" : "FC049DA8FAE3D6B2638B70DFA7471396"
      },
      {
         "nodeId" : 65535,
         "authenticated" : false,
         "linkKey" : "FC049DA8FAE3D6B2638B70DFA7471396",
         "eui" : "00DB1234567890A0"
      }
   ]
}

gfi@gfi:/opt/gfi$
```

**Figure 39 – Control Port – Get Key Table command.**

### 4.4.3 Add Device

Requests the specified device is added to the Communications Hub's Device Log. For this command, the format of the installation credentials parameter is important, and must have spaces between each byte.

```
gfi@gfi:/opt/gfi$ echo \
> {\"command\":\"addDevice\", \
>  \"parameters\":{\"eui\":\"0022A30000020C16\" , \
>                  \"installationCredentials\":\"83 FE D3 40 7A 93 97 23 A5 C6 39 B2 69 16 D5 10\" , \
>                  \"deviceType\" : \"1\" } } | nc localhost 50000
{"action": "addDevice"}
gfi@gfi:/opt/gfi$
```

**Figure 40 – Control Port – Add Device command.**

### 4.4.4 Remove Device

Requests the specified device is removed from the Communications Hub's Device Log.

```
gfi@gfi:/opt/gfi$ echo {\"command\":\"removeDevice\", \"parameters\":{\"eui\":\"0022A30000020C16\" } }
| nc localhost 50000
{"action": "removeDevice"}
gfi@gfi:/opt/gfi$
```

**Figure 41 – Control Port – Remove Device command.**

### 4.4.5 UnJoin Device

Requests the specified device is unjoined from the HAN, this is so that it can perform the join process again.

```
gfi@gfi:/opt/gfi$ echo {\"command\":\"unJoinDevice\", \"parameters\":{\"eui\":\"0022A30000020C16\" } }
| nc localhost 50000
{"action": "unJoinDevice"}
gfi@gfi:/opt/gfi$
```

**Figure 42 – Control Port – UnJoin Device command.**

### 4.4.6 Get Stack Versions

Request the specified device the stack versions that are in use in NCP and in the Host.

```
gfi@gfi:/opt/gfi$ echo '{"command":"getStackVersions"}' | nc localhost 50000
{"stackVersion":{"host":"6.2.3","ncp":"6.2.3"}}gfi@gfi:/opt/gfi$
```

**Figure 43 – Control Port – Get Stack Versions command.**

### 4.4.7 Get CH Time

Request the specified device the UTC  date and time.

```
gfi@gfi:~$ echo '{"command":"getCHTime"}' | nc localhost 50000
{"action":"getCHTime", "date":"07/01/2020" , "time":"14:15"}gfi@gfi:~$
```

**Figure 44 - Control Port - Get CH UTC date and time**

### 4.4.8 Sync CH Time with Operative System

Request the specified device to synchronize the date and time with the Operative System.

```
gfi@gfi:~$ echo '{"command":"syncTimeWithOS"}' | nc localhost 50000
{"action":"syncTimeWithOS"}gfi@gfi:~$
```

**Figure 45 - Control Port - Synchronize CH date and time with Operating System**

### 4.4.9 Set CH Time

Request the specified device to set a specified date and time (UTC).

```
gfi@gfi:~$ echo '{"command":"setCHTime", "parameters":{"date":"10/02/2020", "time":"10:00"}}' | nc localhost 50000
{"action":"setCHTime"}gfi@gfi:~$
```

**Figure 46 - Control Port - Set CH UTC date and time**

### 4.4.10 CH Soft Stop

Request the specified device to stop saving without cleaning previous HAN configuration.

```
gfi@gfi:~$ echo '{"command":"soft-stop"}' | nc localhost 50000
{"action": "soft-stop"}gfi@gfi:~$
```

Figure 47 - Control Port - CH Soft Stop

### 4.4.11 CH Hard Stop

Request the specified device to stop cleaning previous HAN configuration.

```
gfi@gfi:~$ echo '{"command":"hard-stop"}' | nc localhost 50000
{"action": "hard-stop"}gfi@gfi:~$
```

Figure 48 - Control Port - CH Hard Stop

### 4.4.12 Get Zigbee network info

Request the Zigbee network current configuration.

```
gfi@gfi:~$ echo '{"command":"getZigbeeInfo"}' | nc localhost 50000
{ "zigbeeInfo": { "subGhzChannel": 0, "subGhzPage": 28, "ghzChannel": 11,
"nwk": "22222222222222222222222222222222", "panId": 43690, "extendedPanId"
: "000B57FFFEB39336" } }gfi@gfi:~$
```

Notes:

If the values to the subGhzChannel and the subGhzPage are -1, this means that subGhz network was not created.

If the value to the ghzChannel is -1, this means the GHz network was not created.

### 4.4.13 Get CH device info

Request the GPF and CH Ids.

```
gfi@gfi:~$ echo '{"command":"getDeviceInfo"}' | nc localhost 50000
{ "deviceInfo": { "chfId": "00DB1234567890A2", "gpfId": "000B57FFFEB39336"
 } }gfi@gfi:~$
```

# 5 Test Execution

Each output artefact is described in each of the following Sections. A general overview will be presented followed by a detailed description of each of its components.

After a fresh installation of the Operating System provided as part of the GFI tool, the Tester can find the GFI testing tool at /opt/gfi/.

## 5.1 Command Line Execution

As briefly stated before, for a Test to be executed the GFI testing tool application should be invoked in the command line, using the Scenario Configuration file and the Test Properties file as input parameters. The most common invocation command line is presented in Figure 49. After the testing tool binary filename, two mandatory arguments must follow. The first one is the Scenario Configuration file and the second one is the Test Properties file - the Test to be executed (in this example, Use Case ECS40) in an emulated Scenario.

```
$ ./gfi-testing-tool conf/gfi-scenario-emulators.xml tests/ATG-TEST-ECS40.xml
```

**Figure 49 – GFI testing tool command line execution call.**

When the Tester starts the Test execution, the console will show in real time the relevant actions performed in the Test, with the configured verbose level as presented in Figure 50. The default verbose level is set to 4, meaning that all the core, high and medium criticality information will be outputted.

```
~/workspace/atg-testing-tool $ ./gfi-testing-tool conf/gfi-scenario-emulators.xml tests/ATG-TEST-ECS40.xml
Found 1º argument. Searching for file [conf/gfi-scenario-emulators.xml].
Found 2º argument. Searching for file [tests/ATG-TEST-ECS40.xml].


-----------------------------------------------------------------------------------------------
Initializing the Software Database (SDB)...
Starting GFI Execution...
-----------------------------------------------------------------------------------------------
 STARTING EXECUTION TIME: 2015-06-23 21:10:40 (WEST)
-----------------------------------------------------------------------------------------------
    TIME      | AREA  | LV | MESSAGE
-----------------------------------------------------------------------------------------------
00:00:00.138 |  MNGE |  0 | Preparing Test Execution...
00:00:00.138 | RPORT |  4 |     Report location: /home/gpgouveia/workspace/atg-testing-tool/tests/test-report.html
00:00:00.142 |   ACB |  4 |     ACBEmulator: Start processing thread for messages received from the KRP
00:00:00.142 |   ACB |  4 |     ACBEmulator: Waiting for new messages from KRP
00:00:00.143 |   ACB |  4 |     ACBEmulator: Start processing thread for messages received from xSME
00:00:00.143 |   ACB |  4 |     ACBEmulator: Waiting for new messages from xSME
00:00:00.144 | EQUIP |  4 |     CHFEmulator: Start processing thread
00:00:00.144 | EQUIP |  4 |     CHFEmulator: Waiting for new messages
00:00:00.145 | EQUIP |  4 |     ESMEEmulator: Start processing thread
00:00:00.145 | EQUIP |  4 |     ESMEEmulator: Waiting for new messages
00:00:00.146 | EQUIP |  4 |     GSMEEmulator: Start processing thread
00:00:00.146 | EQUIP |  4 |     GSMEEmulator: Waiting for new messages
00:00:00.157 |  MNGE |  0 | Starting Test Execution...
00:00:00.157 |  MNGE |  0 | Starting Run 1 of 1...
00:00:00.157 |  MNGE |  0 | Executing Run 1 of 1...
00:00:00.157 |  MNGE |  2 |   Testcase: ATG-TC-ECS40, Starting...
00:00:00.160 |  MNGE |  2 |   Testcase: ATG-TC-ECS40, Starting Iteration: 1 of 2
00:00:00.161 |  MNGE |  2 |   Testcase: ATG-TC-ECS40, Iteration: 1 of 2, Starting Test Step: 1 of 1
00:00:00.164 |  TEST |  4 |     CGBCSUseCase: Preparing to send GENERAL_SIGNING_MESSAGE
00:00:00.164 |  TEST |  4 |     CGBCSUseCase: Adding signature to GENERAL_SIGNING_MESSAGE
00:00:00.165 | EQUIP |  4 |     KRP: GENERAL_SIGNING_MESSAGE detailed byte array (502 bytes):
00:00:00.165 | EQUIP |  4 |     DF 09 01 00 00 00 00 00 00 05 14 08 00 DB 12 34 56 78 00 01 08 00 DB 12 34 56 78 90 A0 00 82 01
00:00:00.165 | EQUIP |  4 |     B5 00 4E 00 DB 12 34 56 78 00 02 00 00 00 00 00 00 03 E8 30 82 01 9F 30 82 01 46 A0 03 02 01 02
00:00:00.165 | EQUIP |  4 |     02 10 41 52 A9 D6 B9 27 A7 2E CB 05 2A B9 44 12 FA 48 30 0A 06 08 2A 86 48 CE 3D 04 03 02 30 1A
```

**Figure 50 – GFI testing tool execution of a Test: console output overview.**

When the Test execution is complete, an overall PASS/FAIL result will be presented as shown in Figure 51. In the same path where the Test Properties file is located the following outputs will be created:

| Output Artefact | Description |
|---|---|
| test-report.xml | The last successfully Test execution report produced by the tool, in XML format. Every time a Test is executed this file is deleted and a new one created. |
| test-report.html | The last successfully Test execution report produced by the tool, in HTML format. Every time a Test is executed this file is deleted and a new one created. |
| execution-log.csv | The last successfully Test execution log output in comma-separated values format. Every time a Test is executed this file is deleted and a new one created. |
| reports/<execution-file>-test-report-<date>.xml | The <execution-file> test report executed in at <date>, in XML format. This file will never be deleted by the testing tool. |
| reports/<execution-file>-test-report-<date>.html | The <execution-file> test report executed in at <date>, in HTML format. This file will never be deleted by the testing tool. |
| logs/<execution-file>-execution-log-<date>.csv | The <execution-file> execution log output in comma-separated values format. This file will never be deleted by the testing tool. |

**Table 18 – Produced output artefacts upon a Test execution completes.**

```
00:00:00.292 |  MNGE |  2 |   Testcase: ATG-TC-ECS40, Overall Result = PASS
00:00:00.292 |  MNGE |  0 | Complete Run 1 of 1
00:00:00.292 |  MNGE |  0 | Complete Test Execution
00:00:00.292 | RPORT |  0 | ----------------------------------------------------------------
00:00:00.292 | RPORT |  0 | OVERALL TEST RESULT: PASS
00:00:00.292 | RPORT |  0 | ----------------------------------------------------------------
```

**Figure 51 – GFI testing tool overall result overview.**

## 5.2 Command Line Options

Besides the common command line execution, some command line options are available for the Tester to use. The following sections describes the available command line options, their usage and invocation.

### 5.2.1 Help

The command line option “--help” displays a help menu, informing the Tester about the mandatory and optional input parameters. Also a short description of each one of the command line options is provided. Figure 52 presents the invocation of the tool using this command line option and the corresponding output.

```
~/workspace/atg-testing-tool $ ./gfi-testing-tool --help
Usage: gfi-testing-tool [options] <scenario-file.xml> <test-execution-file.xml>
<scenario-file.xml>       The scenario configuration file
<test-execution-file.xml> The Test execution file
Options:
  --help                  Display this information
  --version               Display testing tool version information
  --dcrypto               Disables the rejection of messages due to invalid MAC or Signature
  --checkmsg              Checks if the cryptographic protections (MAC and/or Signature) of the given message are correct.
                          When using this option, the <xml-test-file> should be replaced by a file with the message to check.
  --loglevel  <level>     Configure the console output log level to <level>.
  --outputdir <dir>       Sets <dir> as the base output directory for logs and reports.
```

**Figure 52 – GFI testing tool invocation using the command line option "--help".**

### 5.2.2 Version

The command line option "--version" displays the software version of the GFI testing tool. Figure 53 presents the invocation of the testing tool using this command line option as well as the corresponding output.

```
/opt/gfi $ ./gfi-testing-tool --version
GFI v1.0 RC2 External Release Candidate (Aug  7 2015 10:20:29)
Copyright (c) Smart DCC Ltd.
```

**Figure 53 – GFI testing tool invocation using the command line option "--version".**

### 5.2.3 Cryptography

In some situations, it can be useful to disable the rejection of messages due to invalid MAC and Signature cryptographic verification. For those situations, the command line option "--dcrypto" may be used. When using this command line option, the GFI testing tool will continue to inform if the MAC and Signature were successfully verified or not, but in case of unsuccessful verification the response message will continue to be accepted as if the cryptographic protections would be outputting status OK. Figure 54 presents the invocation of the tool using this command line option.

```
~/workspace/atg-testing-tool $ ./gfi-testing-tool conf/gfi-scenario-emulators.xml tests/ATG-TEST-ECS40.xml --dcrypto
Found 1º argument. Searching for file [conf/gfi-scenario-emulators.xml].
Found 2º argument. Searching for file [tests/ATG-TEST-ECS40.xml].


-----------------------------------------------------------------------------------------------
Initializing the Software Database (SDB)...
Starting GFI Execution...
-----------------------------------------------------------------------------------------------
 STARTING EXECUTION TIME: 2015-06-23 21:15:54 (WEST)
-----------------------------------------------------------------------------------------------
    TIME      | AREA  | LV | MESSAGE
-----------------------------------------------------------------------------------------------
00:00:00.130 |  MNGE |  0 | Preparing Test Execution...
00:00:00.130 | RPORT |  4 |     Report location: /home/gpgouveia/workspace/atg-testing-tool/tests/test-report.html
00:00:00.133 |   ACB |  4 |     ACBEmulator: Start processing thread for messages received from xSME
00:00:00.133 |   ACB |  4 |     ACBEmulator: Waiting for new messages from xSME
00:00:00.135 | EQUIP |  4 |     CHFEmulator: Start processing thread
00:00:00.135 | EQUIP |  4 |     CHFEmulator: Waiting for new messages
00:00:00.136 |   ACB |  4 |     ACBEmulator: Start processing thread for messages received from the KRP
00:00:00.136 |   ACB |  4 |     ACBEmulator: Waiting for new messages from KRP
00:00:00.136 | EQUIP |  4 |     ESMEEmulator: Start processing thread
00:00:00.136 | EQUIP |  4 |     ESMEEmulator: Waiting for new messages
00:00:00.139 | EQUIP |  4 |     GSMEEmulator: Start processing thread
00:00:00.139 | EQUIP |  4 |     GSMEEmulator: Waiting for new messages
00:00:00.148 |   KRP |  2 |   WARNING: Rejection of messages due to invalid MAC or Signature was disabled by User.
00:00:00.148 |  MNGE |  0 | Starting Test Execution...
```

**Figure 54 – GFI testing tool invocation using the command line option "--dcrypto".**

When calling the tool with this command line option, a warning is presented, informing the Tester that the rejection of messages due to invalid MAC and Signature was disabled by the Tester.

### 5.2.4 Offline Check

The command line option “--checkmsg” allows the Tester to check offline the cryptographic protections’ status of a previously received response message. The invocation of the testing tool will differ a little bit from the usual. Figure 56 presents the invocation of the tool using this command line option.

Instead of the Test Properties XML file, a text file (with any extension) containing the response message to be checked should be used as the second input parameter. Figure 55 presents the contents of the file “ECS40-response.txt” used as an input in the example bellow.

```
~/workspace/atg-testing-tool $ cat ECS40-response.txt
DD 00 00 00 00 00 00 6C 11 00 00 00 00 DF 09 02 00 00 00 00 00 00 05 15 08 00 DB 12 34 56 78 90
A0 08 00 DB 12 34 56 78 00 01 00 02 00 4E 38 DA 20 00 05 15 00 00 02 09 1A 31 33 31 32 33 34 35
36 37 38 31 31 31 31 33 31 32 33 34 35 36 37 38 32 32 32 09 0D 31 33 31 32 33 34 35 36 37 38 33
33 33 02 01 00 01 00 00 B6 67 D3 1D 72 81 F0 82 95 9A 61 9C
```

Figure 55 – Contents of the file with the response message to be checked.

```
~/workspace/atg-testing-tool $
~/workspace/atg-testing-tool $ ./gfi-testing-tool conf/gfi-scenario-emulators.xml ECS40-response.txt --checkmsg
Found 1º argument. Searching for file [conf/gfi-scenario-emulators.xml].
Found 2º argument. Searching for file [ECS40-response.txt].


-----------------------------------------------------------------------------------------------
Initializing the Software Database (SDB)...
Starting GFI Execution...
-----------------------------------------------------------------------------------------------
 STARTING EXECUTION TIME: 2015-06-23 21:23:52 (WEST)
-----------------------------------------------------------------------------------------------
    TIME      | AREA  | LV | MESSAGE
-----------------------------------------------------------------------------------------------
00:00:00.121 | CRYPT |  4 |     CryptoChecker: GENERAL_CIPHERING_MESSAGE detailed byte array (132 bytes):
00:00:00.121 | CRYPT |  4 |     DD 00 00 00 00 00 00 7C 11 00 00 00 00 DF 09 02 00 00 00 00 00 00 05 14 08 00 DB 12 34 56 78 90
00:00:00.121 | CRYPT |  4 |     A0 08 00 DB 12 34 56 78 00 01 00 12 00 4E 00 DB 12 34 56 78 00 02 00 00 00 00 00 00 03 E8 38 DA
```

Figure 56 – GFI testing tool invocation using the command line option “--checkmsg”.

At the end of the execution, a PASS/FAIL result is provided as shown in Figure 57.

```
00:00:00.125 | CRYPT |  6 |       KDF Symmetric Key (16 bytes):
00:00:00.125 | CRYPT |  6 |       AA 1B 51 16 E7 A5 38 65 25 23 21 4D 68 53 C9 7D
00:00:00.125 |   KRP |  4 |     RPEmulator: MAC verification succeeded.
00:00:00.125 |   KRP |  4 |     RPEmulator: Signature verification succeeded (empty as expected)
00:00:00.125 | CRYPT |  4 |     CryptoChecker: Message accepted.
00:00:00.125 |  MNGE |  0 | -----------------------------------------------------------------
00:00:00.125 |  MNGE |  0 | OVERALL TEST RESULT: PASS
00:00:00.125 |  MNGE |  0 | -----------------------------------------------------------------
-----------------------------------------------------------------------------------------------
Complete GFI Execution.
-----------------------------------------------------------------------------------------------
```

Figure 57 – Result of the cryptographic verification using the command line option “--checkmsg”.

### 5.2.5 Verbose Level

The command line option “--loglevel” can be used to override the verbose level of the Console Output defined in the Scenario Configuration file. This parameter defines the extent to which the verbosity is detailed. The higher level is defined, the greater amount of information (and less critical) is included in the logging process. If a log level 4 is defined, all log entries marked with levels 0 through 4 will be logged, leaving the remaining (greater than 4) unregistered.

Figure 58 presents the invocation of the tool using this command line option. Table 2 presents the available options for the verbose level as well as a description about the meaning of the value and recommended usage.

```
$ ./gfi-testing-tool conf/gfi-scenario-emulators.xml tests/ATG-TEST-ECS40.xml --loglevel 6
ario-emulators.xml].
T-ECS40.xml].
```

Figure 58 – GFI testing tool invocation using the command line option "--loglevel".

This parameter defines the extent to which the verbosity is detailed. The higher level is defined, the greater amount of information (and less critical) is included in the logging process. If a log level 4 is defined in the Scenario Configuration file (Section 3.1.1), all log entries marked with levels 0 through 4 will be logged, leaving the remaining (greater than 4) unregistered.

### 5.2.6 Output Directory

The command line option "--outputdir" allows the Tester to specify an output path. By default the GFI testing tool will create all the outputs (log and reports) in the same path of the Test Properties XML file. With this option the outputs will not be created in the default location but instead in the path provided. This path can be either absolute or relative, as long as the Tester has permissions to write in the specified folder, the outputs will be correctly produced. If the path doesn't exist, the GFI tool creates all the necessary folders.

Figure 59 presents the invocation of the tool using this command line option.

```
$ ./gfi-testing-tool conf/gfi-scenario-emulators.xml tests/ATG-TEST-ECS40.xml --outputdir TESTLOCATION-1
ario-emulators.xml].
T-ECS40.xml].
```

Figure 59 – GFI testing tool invocation using the command line option "--outputdir".

## 5.3 Test Report: XML

This Section describes the Test Report file referred to as <test-report>.xml. This file contains the raw information from the Test execution, including the input parameters (sets), the computed output parameters along with the expected results (checks) and all the exchanged messages between the framework and the device under test. Figure 60 presents an overview of this file.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<?xml-stylesheet type='text/xsl' href='atg_vsis-execution-report-xsl.xml'?>
<report>

  <test_spec>
    <properties>
      <item name="USE CASE">ECS22a</item>
      <item name="TEST ID">ATG-TEST-ECS22a</item>
      <item name="TEST PURPOSE">Test Use Case ECS22a "Read Half Hour Profile Data (export)", which is for reading the half ho
      <item name="EXECUTION DATE">2015-04-29 18:10:28 (WEST)</item>
    </properties>
    <overall_result>PASS</overall_result>
    <summary/>
    <test_case name="ATG-TC-ECS22a" run="1">
      <iterations>2</iterations>
      <steps>1</steps>
      <properties>
        <item name="TEST CASE ID">ATG-TC-ECS22a</item>
        <item name="TEST CASE PURPOSE">The purpose of this test case is to validate Use Case ECS22a "Read Half Hour Profile D
        <item name="TEST CASE ASSUMPTIONS AND CONSTRAINTS">N/A</item>
        <item name="AUTHOR">CSW</item>
        <item name="VERSION">02</item>
      </properties>
      <execution>
        <iteration name="Test Case Setup"/>
        <iteration name="1">
          <inputs>
            <input name="BusinessOriginatorName">ACB</input>
            <input name="BusinessTargetName">ESME1</input>
            <input name="SupplementaryRemotePartyName">Supplier1</input>
            <input name="ProfileDataLogExport.fromDateTime">2015/01/01 09:30:00</input>
            <input name="ProfileDataLogExport.toDateTime">2015/01/02 09:30:00</input>
            <input name="ProfileDataLogExport.result">0</input>
          </inputs>
          <step name="Iteration Setup"/>
          <step name="1">
            <action actual="0x0000DB000000A013" msg="GENERAL_SIGNING_MESSAGE.GROUPING_HEADER.OTHER_INFORMATION.SUPPLEMENTARY_
            <action actual="0x00000000000003E8" msg="GENERAL_SIGNING_MESSAGE.GROUPING_HEADER.OTHER_INFORMATION.SUPPLEMENTARY_
            <action actual="30 82 01 9F 30 82 01 46 A0 03 02 01 02 02 10 41 52 A9 D6 B9 27 A7 2E CB 05 2A B9 44 12 FA 48 30 6
```


**Figure 60 – Test Report XML file overview.**

The following sections will detail each of the elements in this file.

### 5.3.1 Properties

This heading element presents the information of each respective item in the Test Properties file (Section 3.2.1) plus the execution time and date.


```xml
<test_spec>
  <properties>
    <item name="USE CASE">ECS22a</item>
    <item name="TEST ID">ATG-TEST-ECS22a</item>
    <item name="TEST PURPOSE">Test Use Case ECS22a "Read Half Hour Profile Data (export)", which is for readi
    <item name="EXECUTION DATE">2015-04-29 18:10:28 (WEST)</item>
  </properties>
```


**Figure 61 – Raw report properties.**

### 5.3.2 Overall Result

This element presents the Test overall PASS/FAIL result. It will be a FAIL should any Test Case on any iteration fail, otherwise the overall result will be a PASS.


```xml
</properties>
<overall_result>PASS</overall_result>
<summary/>
```


**Figure 62 – Raw report overall result.**

### 5.3.3 Test Case

#### Iterations and Steps

These two elements define the number of iterations on the Test Case and the number of steps for the iteration.

```
<test_case name="ATG-TC-ECS22a" run="1">
  <iterations>2</iterations>
  <steps>1</steps>
  <properties>
```

Figure 63 – Raw report iterations and steps overview.

#### Properties

Holds the Test Case information gathered from the Test Properties file (Section 3.2.1). Each Test Case block contains one of these elements.

```
<steps>1</steps>
<properties>
  <item name="TEST CASE ID">ATG-TC-ECS22a</item>
  <item name="TEST CASE PURPOSE">The purpose of this test case is to validate Use Case ECS22a "Read Half Hour Profile Data (export)"
  <item name="TEST CASE ASSUMPTIONS AND CONSTRAINTS">N/A</item>
  <item name="AUTHOR">CSW</item>
  <item name="VERSION">02</item>
</properties>
<execution>
```

Figure 64 – Raw report Test Case properties overview.

#### Execution - Iterations and Steps

This nested element presents the execution information. It will nest as many iterations as shown in the element described in Section 5.3.3 which, in its turn, will nest the Inputs information and the Steps information (also as many as shown in section 5.3.3).

#### Inputs

These test inputs refer to the resource pool entries in the Test Properties file (Section 3.2.3). Some of these elements will be used as input parameters and others as expected values to perform checks on the Test Case outputs.

```
<iteration name="1">
  <inputs>
    <input name="BusinessOriginatorName">ACB</input>
    <input name="BusinessTargetName">ESME1</input>
    <input name="SupplementaryRemotePartyName">Supplier1</input>
    <input name="ProfileDataLogExport.fromDateTime">2015/01/01 09:30:00</input>
    <input name="ProfileDataLogExport.toDateTime">2015/01/02 09:30:00</input>
    <input name="ProfileDataLogExport.result">0</input>
  </inputs>
```

Figure 65 – Raw report iteration inputs overview.

#### Steps

Contain the actions taken in an iteration step. As referred before, they will be in the number defined in Section 5.3.3, plus two extra steps (Iteration Setup and Iteration Teardown) in case of any Setup or Clean-up is needed in said iteration.

```
<step name="1">
  <action actual="0x0000DB000000A013" msg="GENERAL_SIGNING_MESSAGE.GROUPING_HEADER.OTHER_INFORMATION.
  <action actual="0x00000000000003E8" msg="GENERAL_SIGNING_MESSAGE.GROUPING_HEADER.OTHER_INFORMATION.
  <action actual="30 82 01 9F 30 82 01 46 A0 03 02 01 02 02 10 41 52 A9 D6 B9 27 A7 2E CB 05 2A B9 44
  <action actual="0xDF" msg="GENERAL_SIGNING_MESSAGE.GROUPING_HEADER.TAG" time="00:00:00.152" type="S
  <action actual="9" msg="GENERAL_SIGNING_MESSAGE.GROUPING_HEADER.TRANSACTION_ID.LENGTH" time="00:00:
  <action actual="0x0000000000000514" msg="GENERAL_SIGNING_MESSAGE.GROUPING_HEADER.TRANSACTION_ID.VAL
  <action actual="8" msg="GENERAL_SIGNING_MESSAGE.GROUPING_HEADER.ORIGINATOR_SYSTEM_TITLE.LENGTH" tim
  <action actual="0x0000DB000000A018" msg="GENERAL_SIGNING_MESSAGE.GROUPING_HEADER.ORIGINATOR_SYSTEM_
  <action actual="8" msg="GENERAL_SIGNING_MESSAGE.GROUPING_HEADER.RECIPIENT_SYSTEM_TITLE.LENGTH" time
  <action actual="0x0022A30000021430" msg="GENERAL_SIGNING_MESSAGE.GROUPING_HEADER.RECIPIENT_SYSTEM_T
  <action actual="0" msg="GENERAL_SIGNING_MESSAGE.GROUPING_HEADER.DATE_TIME.LENGTH" time="00:00:00.15
  <action actual="1" msg="GENERAL_SIGNING_MESSAGE.GROUPING_HEADER.TRANSACTION_ID.VALUE.CRA_FLAG" time
  <action actual="0x0036" msg="GENERAL_SIGNING_MESSAGE.GROUPING_HEADER.OTHER_INFORMATION.MESSAGE_CODE
  <action actual="0x0514" msg="ECS22A_COMMAND.PAYLOAD.LONG_INVOKE_ID_PRIORITY.INVOKE_ID" time="00:00:
  <action actual="0b0000" msg="ECS22A_COMMAND.PAYLOAD.LONG_INVOKE_ID_PRIORITY.RESERVED" time="00:00:6
```

**Figure 66 – Raw report steps overview.**

### Actions and Checks

These are the test actions taken in the test. There are several types, namely:

- "SET" to fill in a field in the command message;
- "SEND" to issue a command message;
- "ACTION" a generic type most commonly used to include waiting times;
- "RECEIVE" to store a response message;
- "GET" to obtain a response message field value for the purpose of Test Case logic (e.g. array field size to use in iterating checks);
- "PRINT" to perform manual check when automatic checks cannot be implemented and to include an entire exchanged message in the report.

The attributes in these elements are:

- actual="..." for the actual value;
- msg="..." to specify the respective message;
- time="…" to include a timestamp for the action;
- type="…" to select from the previously explained different types of actions.

```
  <check actual="0" comparison="equal to" description="ECS22A_RESPONSE.PAYLOAD.ACCESS_RESPONSE_BODY.ACCESS_REQUEST_SPECIFICATION.NUM
  <check actual="1" comparison="equal to" description="ECS22A_RESPONSE.PAYLOAD.ACCESS_RESPONSE_BODY.ACCESS_RESPONSE_LIST_OF_DATA.NUM
  <check actual="1" comparison="equal to" description="ECS22A_RESPONSE.PAYLOAD.ACCESS_RESPONSE_BODY.ACCESS_RESPONSE_SPECIFICATION.NU
  <check actual="0x0001" comparison="equal to" description="ECS22A_RESPONSE.PAYLOAD.ACCESS_RESPONSE_BODY.ACCESS_RESPONSE_SPECIFICATI
  <action actual="49" msg="ECS22A_RESPONSE.PAYLOAD.ACCESS_RESPONSE_BODY.ACCESS_RESPONSE_LIST_OF_DATA.REQUEST_RESPONSES.PROFILE_DATA_
  <action actual="[&#xA;  {&#xA;    &quot;timestamp&quot;:&quot;2015/01/01 09:30:00&quot;,&#xA;    &quot;activeEnergyExportedValue&q
  <check actual="0" comparison="equal to" description="ECS22A_RESPONSE.PAYLOAD.ACCESS_RESPONSE_BODY.ACCESS_RESPONSE_SPECIFICATION.RE
</step>
<step name="Iteration Teardown"/>
</iteration>
```

**Figure 67 – Raw report actions and checks overview.**

As for the checks there is only one type available:

- "CHECK" to perform a verification on a response message field against a given expected result.

This one requires a few more attributes than the ones already described in the Actions:

- comparison="..." for the type of check (e.g. equality, inequity, etc.);
- description="..." information about the Check (typically to discriminate the parameter being checked);
- expected="…" to print the expected result (the actual result will can be found in actual="...");
- result="…" to print the comparison outcome (PASS / FAIL).

### Test Summary

One for each Test Case, contains some statistics (checks performed and passed) and the Test Case PASS/FAIL status.


```
      </execution>
      <test_summary>
        <checks_performed>46</checks_performed>
        <checks_passed>46</checks_passed>
        <test_result>PASS</test_result>
      </test_summary>
    </test_case>
  </test_spec>

</report>
```


**Figure 68 – Raw report summary overview.**

## 5.4 Test Report: HTML

This Section describes the Test Report file referred to as <test-report>.html

**GFI-TESTING-TOOL TEST REPORT**

TEST INFORMATION

| | |
|---|---|
| **Test Name** | ECS40 |
| **Test Id** | ATG-TEST-ECS40 |
| **Test Purpose** | Test Use Case ECS40 "Read MPAN Value on the ESME", which is for reading the MPAN on the ESME. |
| **Execution Date** | 2015-06-30 11:52:12 (WEST) |
| **Overall Result** | PASS |

TEST CASES RESULTS

| Test Case | Result |
|---|---|
| ATG-TC-ECS40 | PASS |

TEST CASES EXECUTION

| | |
|---|---|
| **ATG-TC-ECS40** | |
| **Test Case Information** | − |
| **Test Case Id** | ATG-TC-ECS40 |
| **Test Case Purpose** | The purpose of this test case is to validate Use Case ECS40 "Read MPAN Value on the ESME" by validating the construction of the command used for reading the power registers on the ESME and its respective response. |
| **Test Case Assumptions And Constraints** | GIST Issue #3145: "ECS40 - Notes for attribute MeterPointAdministrationNumber(combinedMPANS)" |
| **Author** | CSW |
| **Version** | 01 |
| **Iteration 1** | − |
| **Inputs** | + |
| **Execution** | + |
| **Results Summary** | − |
| **Overall Iteration Expected Results Matched 100% (30/30)** | |
| **Result** | PASS |
| **Iteration 2** | − |
| **Inputs** | + |
| **Execution** | + |
| **Results Summary** | − |
| **Overall Iteration Expected Results Matched 100% (30/30)** | |
| **Result** | PASS |
| **Test Case Results Summary** | − |
| **Test Case Expected Results Matched 100% (60/60)** | |
| **Result** | PASS |

Multi-purpose System Validation Facility - Copyright (c) Smart DCC Ltd.

**Figure 69 – Test Report (Formatted - html)**

This file contains the formatted information from the Test Properties, including the input parameters (sets), the computed output parameters along with the expected results (checks) and all the exchanged messages between the framework and the device under test.

In the following subsections a description of this report will be presented along with some references to the XML elements used to generate each component.

### 5.4.1 Test Information

This is the Test procedure information, it presents the information of each respective item in the Test Properties file (Section 3.2.1) plus the execution time and date. It also contains the Overall Result information.

TEST INFORMATION

| | |
|---|---|
| **Test Name** | ECS40 |
| **Test Id** | ATG-TEST-ECS40 |
| **Test Purpose** | Test Use Case ECS40 "Read MPAN Value on the ESME", which is for reading the MPAN on the ESME. |
| **Execution Date** | 2015-06-30 11:52:12 (WEST) |
| **Overall Result** | PASS |

**Figure 70 – Test Information**

### 5.4.2 Test Cases Results

The PASS/FAIL result of each Test Case gathered from the collection of checks in the XML elements described in Section 5.3.3 (Test Summary). This example contains only one test case but a procedure with N test cases will produce N entries in this report section showing the result for each test case.

TEST CASES RESULTS

| Test Case | Result |
|---|---|
| ATG-TC-ECS40 | PASS |

**Figure 71 – Test Case Results**

### 5.4.3 Test Cases Execution

For each test case in a Test procedure, a group such as this one (following subsections) will be printed in the report. Like the previous sections, contains the formatted information from the raw test report described in Section 5.3.3 and subsections. Namely:

#### Test Case Information

Holds the Test Case information gathered from the Test Properties file (Section 3.2.1). Each Test Case block contains one of these elements.

This report section contains the information of a given test case See Section 5.3.3 (Properties).

TEST CASES EXECUTION

| ATG-TC-ECS40 | |
|---|---|
| **Test Case Information** | − |
| **Test Case Id** | ATG-TC-ECS40 |
| **Test Case Purpose** | The purpose of this test case is to validate Use Case ECS40 "Read MPAN Value on the ESME" by validating the construction of the command used for reading the power registers on the ESME and its respective response. |
| **Test Case Assumptions And Constraints** | GIST Issue #3145: "ECS40 - Notes for attribute MeterPointAdministrationNumber(combinedMPANS)" |
| **Author** | CSW |
| **Version** | 01 |
| **Iteration 1** | − |
| **Inputs** | + |

**Figure 72 – Test Case Information**

#### Iteration Inputs

These test inputs refer to the resource pool entries in the Test Properties file (Section 3.2.3). Some of these elements will be used as input parameters and others as expected values to perform checks on the Test Case outputs. This is the formatted information from Section 5.3.3 (Inputs).

| | |
|---|---|
| Version | 01 |
| **Iteration 1** | |
| **Inputs** | |
| **BusinessOriginatorName** | ACB |
| **BusinessTargetName** | ESME1 |
| **SupplementaryRemotePartyName** | Supplier1 |
| **MeterAdministrationPointNumber(combinedMPANS).value** | 131234567811113123456782 22 |
| **MeterAdministrationPointNumber(exportMPAN).value** | 1312345678333 |
| **MeterAdministrationPointNumber(combinedMPANS).result** | 0 |
| **MeterAdministrationPointNumber(exportMPAN).result** | 0 |
| **Execution** | |
| **Results Summary** | |

**Figure 73 – Iteration Inputs**

## Iterations Execution

This report section presents the execution information (formatted XML report section described in Section 5.3.3 (Execution – Iterations and Steps)). These entries include all the actions, exchanged messages and performed checks (further information in Section 5.3.3 (Actions and Checks)).

**Execution**

| TIME (HH:MM:SS) | ACTION | DESCRIPTION | VALUE | COMPARISON | EXPECTED | RESULT |
|---|---|---|---|---|---|---|
| 00:00:00.155 | SET | GENERAL_SIGNING_MESSAGE.GROUPING_HEADER.TAG | 0xDF | | | |
| 00:00:00.155 | SET | GENERAL_SIGNING_MESSAGE.GROUPING_HEADER.TRANSACTION_ID.LENGTH | 9 | | | |
| 00:00:00.155 | SET | GENERAL_SIGNING_MESSAGE.GROUPING_HEADER.TRANSACTION_ID.VALUE.ORIGINATOR_COUNTER | 0x0000000000000514 | | | |
| 00:00:00.155 | SET | GENERAL_SIGNING_MESSAGE.GROUPING_HEADER.ORIGINATOR_SYSTEM_TITLE.LENGTH | 8 | | | |
| 00:00:00.155 | SET | GENERAL_SIGNING_MESSAGE.GROUPING_HEADER.ORIGINATOR_SYSTEM_TITLE.VALUE | 0x00DB123456780001 | | | |
| 00:00:00.155 | SET | GENERAL_SIGNING_MESSAGE.GROUPING_HEADER.RECIPIENT_SYSTEM_TITLE.LENGTH | 8 | | | |
| 00:00:00.155 | SET | GENERAL_SIGNING_MESSAGE.GROUPING_HEADER.RECIPIENT_SYSTEM_TITLE.VALUE | 0x00DB1234567890A0 | | | |
| 00:00:00.155 | SET | GENERAL_SIGNING_MESSAGE.GROUPING_HEADER.DATE_TIME.LENGTH | 0 | | | |
| 00:00:00.155 | SET | GENERAL_SIGNING_MESSAGE.GROUPING_HEADER.TRANSACTION_ID.VALUE.CRA_FLAG | 1 | | | |
| (...) | | | | | | |
| (...) | | | | | | |
| | | | FF 02 02 00 00 00 | | | |
| 00:00:00.164 | SET | GENERAL_CIPHERING_MESSAGE.MAC | 24 02 E8 94 DF DF 6D A7 6D A5 5B F3 | | | |
| 00:00:00.170 | ACTION | Waiting for response (ECS40_RESPONSE) | | | | |
| 00:00:00.170 | SEND | ACB: GENERAL_CIPHERING_MESSAGE (529 bytes) | DD 00 00 00 00 00 00 82 02 07 11 00 00 00 00 DF 09 01 00 00 00 00 00 00 05 14 08 00 DB 12 34 56 78 00 01 08 00 DB 12 34 56 78 90 A0 00 82 01 B5 00 4E 00 DB 12 34 56 78 00 02 00 00 00 00 00 00 03 E8 30 82 01 9F 30 82 01 46 A0 03 02 01 02 02 10 41 52 A9 D6 B9 27 A7 2E CB 05 2A B9 44 12 FA 48 30 0A 06 08 2A 86 48 CE 3D 04 03 02 30 1A 31 0B 30 09 06 03 55 04 0B 13 02 30 37 31 0B 30 09 06 03 55 04 03 13 02 4F 42 30 1E 17 0D 31 35 30 34 31 35 30 30 30 30 30 30 5A 17 0D 32 35 30 34 31 34 32 33 35 39 35 39 5A 30 21 31 0B 30 09 06 03 55 04 0B 0C 02 30 32 31 12 30 10 06 03 55 04 2D 03 09 00 00 DB 12 34 56 78 00 02 30 59 30 13 06 07 2A 86 48 CE 3D 02 01 06 08 2A 86 48 CE 3D 03 01 07 03 42 00 04 E0 9B 55 CB F0 75 6B F0 AB 07 4E 70 8C E0 5E 00 31 21 C0 66 7B 94 83 0A 31 E1 86 8B AE 46 3D 58 96 89 31 0C 63 35 10 14 67 40 4F 2E 4C 0F 8D 9B D0 FD DC 4D AB 67 A9 5B DA 75 7A 28 B0 F5 8C A1 A3 67 30 65 30 0C 06 03 55 1D 13 01 01 FF 04 02 30 00 30 0E 06 03 55 1D 0F 01 01 FF 04 04 03 02 03 08 30 11 06 03 55 1D 0E 04 0A 04 08 4F D6 68 AE AA 56 96 47 30 1D 06 03 55 1D 20 01 01 FF 04 13 30 11 30 0F 06 0D 2A 86 3A 00 01 84 8F B9 0F 01 02 01 04 30 13 06 03 55 1D 23 04 0C 30 0A 80 08 4D DF 1E A1 B8 BD 9F BB 30 0A 06 08 2A 86 48 CE 3D 04 03 02 03 47 00 30 44 02 20 1A A6 B5 09 E8 9F 74 46 6B 0B A3 C0 E6 12 0E CC EE 24 F6 E5 B9 E9 52 2C 6E 97 08 25 46 3D 2D 58 02 20 54 C3 C2 C0 70 AC 80 37 31 24 3E 6B 0B 10 17 1B 52 67 FE 1B E2 B2 B7 4D D4 EB 46 1C 6A CF 93 01 1E D9 20 00 05 14 00 02 01 00 01 00 00 5E 2C 22 0B 02 01 00 01 01 00 00 00 02 FF 02 02 00 00 00 24 02 E8 94 DF DF 6D A7 6D A5 5B F3 | | | |
| 00:00:00.181 | RECEIVE | ACB: GENERAL_CIPHERING_MESSAGE (132 bytes) | DD 00 00 00 00 00 00 7C 11 00 00 00 00 DF 09 02 00 00 00 00 00 00 05 14 08 00 DB 12 34 56 78 90 A0 08 00 DB 12 34 56 78 00 01 00 12 00 4E 00 DB 12 34 56 78 00 02 00 00 00 00 00 00 03 E8 38 DA 20 00 05 14 00 00 02 09 1A 31 33 31 32 33 34 35 36 37 38 31 31 31 31 33 31 32 33 34 35 36 37 38 32 32 32 09 0D 31 33 31 32 33 34 35 36 37 38 33 33 33 02 01 00 01 00 00 C6 D7 71 A6 BE 5F E0 46 E8 51 12 E7 | | | |
| (...) | | | | | | |
| (...) | | | | | | |
| | | T_RESULTS.METER_ADMINISTRATION_POINT_NUMBER_COMBINED_MPANS.ACCESS_RESPONSE_TYPE | | | | |
| 00:00:00.187 | CHECK | ECS40_RESPONSE.PAYLOAD.ACCESS_RESPONSE_BODY.ACCESS_RESPONSE_SPECIFICATION.REQUEST_RESULTS.METER_ADMINISTRATION_POINT_NUMBER_EXPORT_MPAN.ACCESS_RESPONSE_TYPE | 1 | equal to | 1 | PASS |
| 00:00:00.187 | CHECK | GENERAL_SIGNING_MESSAGE.GROUPING_HEADER.OTHER_INFORMATION.MESSAGE_CODE | 0x004E | equal to | 0x004E | PASS |
| 00:00:00.187 | CHECK | ECS40_RESPONSE.PAYLOAD.ACCESS_RESPONSE_BODY.ACCESS_RESPONSE_LIST_OF_DATA.REQUEST_RESPONSES.METER_ADMINISTRATION_POINT_NUMBER_COMBINED_MPANS | 13123456781111 312345678222 | equal to | 13123456781111 312345678222 | PASS |
| 00:00:00.187 | CHECK | ECS40_RESPONSE.PAYLOAD.ACCESS_RESPONSE_BODY.ACCESS_RESPONSE_LIST_OF_DATA.REQUEST_RESPONSES.METER_ADMINISTRATION_POINT_NUMBER_EXPORT_MPAN | 1312345678333 | equal to | 1312345678333 | PASS |
| 00:00:00.187 | CHECK | ECS40_RESPONSE.PAYLOAD.ACCESS_RESPONSE_BODY.ACCESS_RESPONSE_SPECIFICATION.REQUEST_RESULTS.METER_ADMINISTRATION_POINT_NUMBER_COMBINED_MPANS.RESULT | 0 | equal to | 0 | PASS |
| 00:00:00.187 | CHECK | ECS40_RESPONSE.PAYLOAD.ACCESS_RESPONSE_BODY.ACCESS_RESPONSE_SPECIFICATION.REQUEST_RESULTS.METER_ADMINISTRATION_POINT_NUMBER_EXPORT_MPAN.RESULT | 0 | equal to | 0 | PASS |

**Results Summary**

**Iteration 2**

**Inputs**

**Execution**

**Figure 74 – Iteration Execution**

### Iteration Results Summary

One for each Iteration containing statistics of checks performed and passed and the Test Case PASS/FAIL status (the PASS/FAIL result of each Iteration's collection of checks). If all the checks in the Iteration pass the summary result will be PASS. If any of these checks fail the summary result will be FAIL.

| | |
|---|---|
| **Results Summary** | |
| **Overall Iteration Expected Results Matched 100% (30/30)** | |
| **Result** | ✔ PASS |
| **Iteration 2** | |
| **Inputs** | |
| **Execution** | |

**Figure 75 – Iteration Results Summary**

## 5.4.4 Test Case Results Summary

One for each Test Case, contains some statistics (checks performed and passed) and the Test Case PASS/FAIL status. See Section 5.3.3 (Test Summary).

| | | | | | | |
|---|---|---|---|---|---|---|
| 00:00:00.220 | CHECK | ECS40_RESPONSE.PAYLOAD.ACCESS_RESPONSE_BODY.ACCESS_RESPONSE_SPECIFICATION.REQUEST_RESULTS.METER_ADMINISTRATION_POINT_NUMBER_EXPORT_MPAN.RESULT | 0 | equal to | 0 | PASS |

| | |
|---|---|
| **Results Summary** | |
| **Overall Iteration Expected Results Matched 100% (30/30)** | |
| **Result** | ✔ PASS |
| **Test Case Results Summary** | |
| **Test Case Expected Results Matched 100% (60/60)** | |
| **Result** | ✔ PASS |

Multi-purpose System Validation Facility - Copyright (c) Smart DCC Ltd.

**Figure 76 – Test Case Results Summary**

## 5.5 Console Execution Output

This Section describes the console output.

```
Found 1º argument. Searching for file [conf/gfi-scenario-emulators.xml].
Found 2º argument. Searching for file [tests/ATG-TEST-EVENT-2-ESME-FULL-RUN.xml].


-----------------------------------------------------------------------------------------------
Initializing the Software Database (SDB)...
Starting GFI Execution...
-----------------------------------------------------------------------------------------------
 STARTING EXECUTION TIME: 2015-06-23 21:32:53 (WEST)
-----------------------------------------------------------------------------------------------
    TIME      | AREA  | LV | MESSAGE
-----------------------------------------------------------------------------------------------
00:00:00.169 |  MNGE |  0 | Preparing Test Execution...
00:00:00.198 |  MNGE |  0 | Starting Test Execution...
00:00:00.198 |  MNGE |  0 | Starting Run 1 of 1...
00:00:00.198 |  MNGE |  0 | Executing Run 1 of 1...
00:00:00.198 |  MNGE |  2 |   Testcase: ATG-TC-CS02a, Starting...
00:00:00.202 |  MNGE |  2 |   Testcase: ATG-TC-CS02a, Starting Iteration: 1 of 2
00:00:00.202 |  MNGE |  2 |   Testcase: ATG-TC-CS02a, Iteration: 1 of 2, Starting Test Step: 1 of 1
00:00:00.235 |  MNGE |  2 |   Testcase: ATG-TC-CS02a, Iteration: 1 of 2, Complete Test Step: 1 of 1
00:00:00.235 |  MNGE |  2 |   Testcase: ATG-TC-CS02a, Complete Iteration: 1 of 2
00:00:00.236 |  MNGE |  2 |   Testcase: ATG-TC-CS02a, Starting Iteration: 2 of 2
00:00:00.236 |  MNGE |  2 |   Testcase: ATG-TC-CS02a, Iteration: 2 of 2, Starting Test Step: 1 of 1
00:00:00.295 |  MNGE |  2 |   Testcase: ATG-TC-CS02a, Iteration: 2 of 2, Complete Test Step: 1 of 1
00:00:00.295 |  MNGE |  2 |   Testcase: ATG-TC-CS02a, Complete Iteration: 2 of 2
00:00:00.295 |  MNGE |  2 |   Testcase: ATG-TC-CS02a, Complete
00:00:00.295 |  MNGE |  2 |   Testcase: ATG-TC-CS02a, Overall Result = PASS
00:00:00.295 |  MNGE |  2 |   Testcase: ATG-TC-CS02b, Starting...
00:00:00.298 |  MNGE |  2 |   Testcase: ATG-TC-CS02b, Starting Iteration: 1 of 2
00:00:00.300 |  MNGE |  2 |   Testcase: ATG-TC-CS02b, Iteration: 1 of 2, Starting Test Step: 1 of 1
00:00:00.883 |  MNGE |  2 |   Testcase: ATG-TC-CS02b, Iteration: 1 of 2, Complete Test Step: 1 of 1
00:00:00.883 |  MNGE |  2 |   Testcase: ATG-TC-CS02b, Complete Iteration: 1 of 2
00:00:00.883 |  MNGE |  2 |   Testcase: ATG-TC-CS02b, Starting Iteration: 2 of 2
00:00:00.884 |  MNGE |  2 |   Testcase: ATG-TC-CS02b, Iteration: 2 of 2, Starting Test Step: 1 of 1
00:00:00.926 |  MNGE |  2 |   Testcase: ATG-TC-CS02b, Iteration: 2 of 2, Complete Test Step: 1 of 1
00:00:00.926 |  MNGE |  2 |   Testcase: ATG-TC-CS02b, Complete Iteration: 2 of 2
00:00:00.926 |  MNGE |  2 |   Testcase: ATG-TC-CS02b, Complete
00:00:00.926 |  MNGE |  2 |   Testcase: ATG-TC-CS02b, Overall Result = PASS
00:00:00.926 |  MNGE |  2 |   Testcase: ATG-TC-CS02c, Starting...
00:00:00.931 |  MNGE |  2 |   Testcase: ATG-TC-CS02c, Starting Iteration: 1 of 3
```

**Figure 77 – CLI Output**

The output console shows the standard output and the error messages of the Test execution in a real time manner. This is useful for monitoring the Test execution. The following Sections will elaborate on the information and the verbose level.

### 5.5.1 Time

The timestamp of a log entry. This is relative to the STARTING EXECUTION TIME, at the beginning of the Test Case execution. It is shown in the format hh:mm:ss.ms.

### 5.5.2 Area

The logging area that the entry refers to. There are a few predefined areas: Management (MNGE), Report (RPORT), Test (TEST) and Equipment (EQUIP).

### 5.5.3 Level

The verbose level associated to the log entry. This could be used for filtering purposes by adjusting the Scenario Configuration file (Section 3.1).

### 5.5.4 Message

Description of the log entry. Information regarding the Test action, check or error message being logged. This information is indented by the number of spaces equivalent to the log level.

## 5.6 Execution Log File

This Section describes the Execution log file referred to as <execution-log>.csv.

```
-----------------------------------------------------------------------------------------------------
 LOG CREATED ON: 2015-06-12 10:29:50 (WEST)
-----------------------------------------------------------------------------------------------------
SEQ NUMBER; TIME (hh:mm:ss.ms); AREA; LEVEL; MESSAGE
1;00:00:00.121;MNGE;6; Created Equipment SimulationManager
2;00:00:00.121;MNGE;6; Created Equipment ACBEmulator
3;00:00:00.122;MNGE;6; Created Equipment RPEmulator
4;00:00:00.122;TDRIV;6; Setting TestDrive context where Sets and Sends will be performed = 4
5;00:00:00.122;TDRIV;6; Setting TestDrive context where Gets and Checks will be performed = 3
6;00:00:00.125;MNGE;6; Created Equipment ESMEEmulator
7;00:00:00.125;TDRIV;6; Setting TestDrive context where Sets and Sends will be performed = 4
8;00:00:00.125;TDRIV;6; Setting TestDrive context where Gets and Checks will be performed = 3
9;00:00:00.128;MNGE;6; Created Equipment GSMEEmulator
10;00:00:00.128;TDRIV;6; Setting TestDrive context where Sets and Sends will be performed = 4
11;00:00:00.128;TDRIV;6; Setting TestDrive context where Gets and Checks will be performed = 3
12;00:00:00.129;MNGE;6; Created Equipment CHFEmulator
13;00:00:00.129;MNGE;6; Created Codec gbcsGeneralCipheringCodec
14;00:00:00.129;MNGE;6; Created Codec gbcsGeneralSigningCodec
15;00:00:00.129;MNGE;6; Created Codec gbcsGBTCodec
16;00:00:00.129;MNGE;6; Created Codec genericCodec
17;00:00:00.129;MNGE;6; Created Codec gbcsGenericArrayCodec
18;00:00:00.129;MNGE;6; Created Codec gbcsVarArrayCodec
19;00:00:00.129;MNGE;6; Created Codec gbcsDateTimeCodec
20;00:00:00.129;MNGE;6; Created Codec gbcsByteArrayCodec
```

**Figure 78 – Execution log overview.**

This file contains a detailed log of the test execution and its verbose detail is customisable through the Scenario Configuration file. Besides the sequencer number, which is only present in the execution log, the information in this file is just about the same as the information buffered through the output of the system console (Section 5.5) but formatted as coma separated values. This allows further execution analysis and manual checks. Although the log level is, as referred, customisable, it is highly recommended that this level is always kept at the maximum available. By default this value is 100. For detailed information on this file's fields, go through the respective subsections of Section 5.5.

### 5.6.1 Sequence number

This is a log entry identifier which is incremented for each new entry. A log entry that exceeds the maximum length is continued in the next line under the same sequence number.

# 6 Use Case Creation/Upgrade

This section is intended for Advanced Users. Knowledge of object-oriented programming and C++ is required. Users should keep this in mind.

Should any new Use Cases be considered for implementation the process defined in this section should be followed. The implemented Use Cases provided with GFI are the ones specified by GBCS v4.0 and should be enough to build any usage scenario. They represent all the actions that may be undertaken while communicating with an ESME or GSME. Nevertheless, should any Use Cases suffer changes in upcoming versions or any new ones arise, the next Sections describe the process to follow, in a step-by-step fashion.

For Use Case upgrading purposes the files described here already exist. In most cases engineering the files from scratch will be harder work than adapting the existing ones. In any case, the process is the same as for creating a new one, with the difference that the files described already exist.

Please note that a new or modified Use Case is meant to be used with a real device and for this reason, such Use Case will not correctly work with GFI emulated devices.

## 6.1 Process Overview

An overview of the process that should be followed in order to create a new Use Case from scratch is presented in Figure 79.

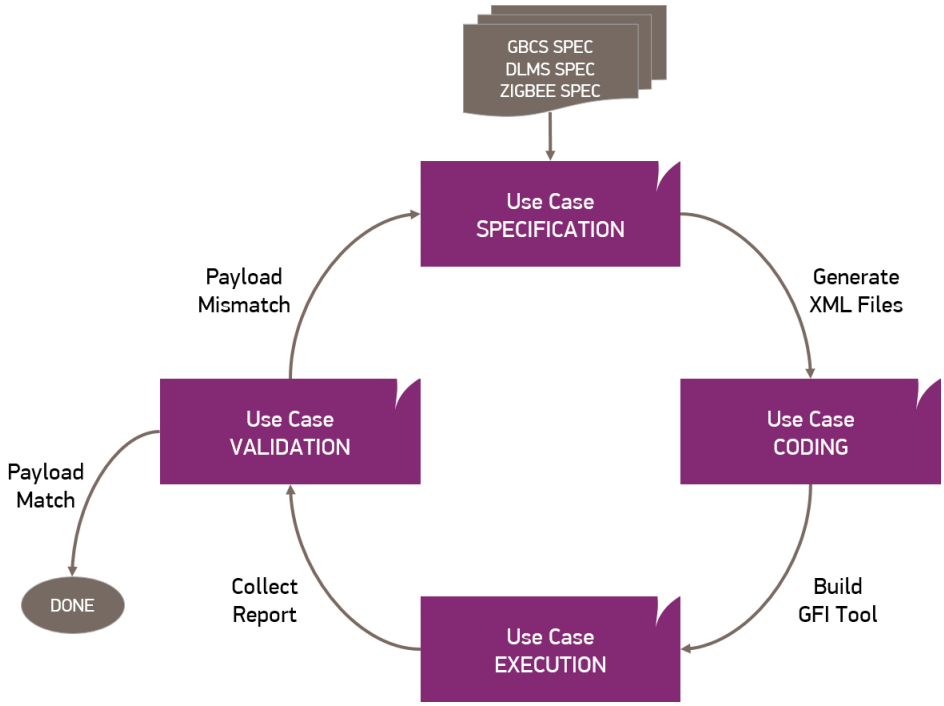


**Figure 79 – Use Case creation process overview.**

The first step (Section 6.2) is to create the Use Case specification, taking as inputs the GBCS, DLMS (Coloured books) and ZigBee (ZCL and ZSE) specifications. In the first step the User must specify: the Command, Response and Alert (if applicable) message payloads; the name and value of the Use Case inputs and expected outputs; and the actions that must be executed in the Test Case.

Once the message payloads, inputs, expected outputs and test case actions are specified, then it is required to generate all the inputs needed for the next step: Use Case Coding. The Use Case specification template provided contains easy mechanisms to produce all the data needed for the Coding phase, namely: the Command Payload XML file, the Response Payload XML file, the Alert Payload XML file and the XML Execution file.

The next step, Use Case Coding (Section 6.3), is the most demanding step as it requires knowledge of object-oriented programming and C++. A set of C++ classes must be produced and coded in order to add the new Use Case to the GFI testing tool Use Case Library. Typically five C++ classes must be produced: Use Case Base class, Use Case Command class, Use Case Response class, Use Case Alert class (if applicable) and Test Case class. Also, the Emulators (ESME, GSME and eventually CHF) C++ implementation must be updated to support the new Use Case. In the emulators, the User typically have to code the Command handler, Response handler and Alert handler (if applicable) methods. After coding

all these classes and methods, the Use Case is ready to be compiled and integrated in the GFI testing tool Library. Once successfully compiled, the Use Case is ready to be executed.

The next step is Use Case Execution (Section 6.4); this is comparatively simple compared to the previous step. The GFI testing tool is executed with the Execution XML file produced in the first step as one of the inputs and a HTML report is produced by the tool. See Section 5 for further details.

The last phase of the process is the Use Case Validation (Section 6.5). The User must check if the Use Case execution succeeded and if all the payloads (Command, Response and Alert) agree with the GBCS, DLMS and ZigBee specification. In the event of all payloads are according to the specifications, the job is done, the new Use Case is now available in the GFI Test Library and ready to be integrated in a Test. If a mismatch is found on the payloads, the User should make the necessary corrections/modifications in the specification and the process starts all over again from the beginning.

The following sections present a more detailed description of each of the four phases briefly presented above.

## 6.2 Use Case Specification

To start, a Use Case Specification document should be produced. This will be used to generate the prerequisites files for the Use Case Coding and Use Case Execution steps. The files generated from the Use Case Specification are as follows:

- The Use Case Command Payload specification file;
- The Use Case Response Payload specification file;
- The Use Case Alert Payload specification file (when applicable);
- The Use Case Execution file.

### 6.2.1 Command Payload

The internal message structures used by the GFI testing tool regarding the command message will be created based on this input. Each element in the message it is tabbed with whitespace, and this should be kept in mind, for it is used for the hierarchy of these same elements in the structure. The data type, codec information and codec encoding are also relevant, as they will be used later by the GFI testing tool to build the internal message structures. As an example, Figure 80 presents an overview of a command payload specification.

```
<structure name="access_request_body" codec="gbcsGenericStructureCodec">
    <structure name="access_request_specification" codec="gbcsGenericStructureCodec">
        <field name="number_of_requests" type="uint8" codec="gbcsIntegerCodec" encoding="uint8"/>
        <structure name="requests" codec="gbcsGenericStructureCodec">
            <structure name="supply_tamper_state_value" codec="gbcsGenericStructureCodec">
                <field name="access_request_type" type="uint8" codec="gbcsIntegerCodec" encoding="uint8"/>
                <structure name="attribute_descriptor" codec="gbcsGenericStructureCodec">
                    <field name="class_id" type="uint16" codec="gbcsIntegerCodec" encoding="uint16"/>
                    <field name="obis_code" type="string" codec="gbcsObisCodeCodec" max_length="24"/>
                    <field name="id" type="uint8" codec="gbcsIntegerCodec" encoding="uint8"/>
                </structure>
            </structure>
        </structure>
    </structure>
    <structure name="access_request_list_of_data" codec="gbcsGenericStructureCodec">
        <field name="number_of_request_parameters" type="uint8" codec="gbcsIntegerCodec" encoding="uint8"/>
        <structure name="request_parameters" codec="gbcsGenericStructureCodec">
            <field name="supply_tamper_state_value" type="uint8" codec="dlmsBooleanCodec" report_format="boolean"/>
        </structure>
    </structure>
</structure>
```

**Figure 80 – Command Payload overview.**

The User should specify the payload structure of the Use Case's command. For each element in the message, the User must specify the data type, the associated codec, the codec encoding, the maximum length of the field (if applicable, meaning the number of bytes) and for bit structures, the bit size, the start bit and the start byte of the element (fields not visible in the figure).

### 6.2.2 Response Payload

Like the Command Payload, the information in the Response Payload is essential. The internal message structures used by the GFI testing tool regarding the response message will be created based on this input. As before, each element in the message it is tabbed with whitespace, and this should be kept in mind, for it is used for the hierarchy of these same elements in the structure.The data type, codec information and codec encoding are also relevant, as they will be used later by the GFI testing tool to build the internal message structures. As an example, Figure 81 presents an overview of a response payload specification.

```
<structure name="access_response_body" codec="gbcsGenericStructureCodec">
    <structure name="access_request_specification" codec="gbcsGenericStructureCodec">
        <field name="number_of_requests" type="uint8" codec="gbcsIntegerCodec" encoding="uint8"/>
    </structure>
    <structure name="access_response_list_of_data" codec="gbcsGenericStructureCodec">
        <field name="number_of_request_responses" type="uint8" codec="gbcsIntegerCodec" encoding="uint8"/>
        <structure name="request_responses" codec="gbcsGenericStructureCodec">
            <field name="supply_tamper_state_value" type="uint8" codec="dlmsNullDataCodec"/>
        </structure>
    </structure>
    <structure name="access_response_specification" codec="gbcsGenericStructureCodec">
        <field name="number_of_request_results" type="uint8" codec="gbcsIntegerCodec" encoding="uint8"/>
        <structure name="request_results" codec="gbcsGenericStructureCodec">
            <structure name="supply_tamper_state_value" codec="gbcsGenericStructureCodec">
                <field name="access_response_type" type="uint8" codec="gbcsIntegerCodec" encoding="uint8"/>
                <field name="result" type="uint8" codec="gbcsIntegerCodec" encoding="uint8"/>
            </structure>
        </structure>
    </structure>
</structure>
```

**Figure 81 – Response Payload overview.**

### 6.2.3 Alert Payload

The format and semantics of the Alert Payload is much the same way as the previous Command and Response Payloads. It defines the internal message structures used by the GFI testing tool regarding the alert message. As an example, Figure 82 presents an overview of an alert payload specification.

```
<structure name="notification_body" codec="dlmsStructureCodec">
    <field name="alert_code" type="uint16" codec="dlmsIntegerCodec" encoding="long-unsigned" report_format="hex04x"/>
    <field name="timestamp" type="datetime" codec="dlmsDateTimeCodec" encoding="octet-string" max_length="12"/>
</structure>
```

**Figure 82 – Alert Payloa overview.**

### 6.2.4 Use Case Test

The Use Case Test Properties file contains, for each iteration, the input values that will be set and the expected values against which the output parameters will be checked. As described in Section 3.2.3, each parameter is composed by one Resource Pool, which contains the input values. To change the number of iterations, simply add or remove the input values in each Resource Pool. Each parameter, Resource Pool, must have the same number of inputs. As an example, Figure 84 presents an overview of one Test Properties with four parameters with one iteration.

#### Internal vs External Parameters

The GFI testing tool differentiates between two types of parameters: internal and external parameters.

The external parameters are used as input values that will be set in the commands or as expected values against which the values in the responses will be checked.

The internal parameters are exclusively used in emulated scenarios and are used to control the behaviour of the GFI Emulators. These internal parameters are sent to Emulators only, using an internal communication channel and a specific protocol. They also follow a rule for the naming, so it is easy to identify them: all start with “ATG.Control” and “Control.”. Some examples of use regarding the emulator’s behaviour control are the following ones:

- Switch between Single and Twin element;
- Set the type of reply on an Event Log reading (reply with empty or with data log);
- Trigger the sending of alerts, namely GNCA, GCA, FDRA, ECS68, …;
- Set the error scenario.

Note that these internal messages are not sent nor received by a physical device.

### 6.2.5 XML Assembling

To create the Payload Specification XML files and the Test Properties XML file use the most similar XML files from GFI testing tool as template, and change them to produce the Command, Response and Alert (when applicable) Payload, and Use Case Execution file.

#### Message’s Payload Specification files

Three XML files containing the specification for the Command, Response and Alert (when applicable) payload messages. These files will subsequently be used to generate the GFI testing tool structures to

support message building, exchanging and encoding/decoding during the Test Case execution. As an example, Figure 83 presents an overview of an assembled Command Payload XML file.

```
<?xml version="1.0" encoding="UTF-8"?>
<structure name="payload" codec="gbcsGenericStructureCodec">

    <field name="tag" type="uint8" codec="gbcsIntegerCodec" encoding="uint8" report_format="xhex"/>
  <structure name="long_invoke_id_priority" codec="gbcsBitStructureCodec">
        <field name="invoke_id"         type="uint32" start_byte="0" start_bit="0" bit_size="24" codec="gbcsBitCo
        <field name="reserved"          type="uint8"  start_byte="3" start_bit="0" bit_size="4"  codec="gbcsBitCo
        <field name="self_descriptive"  type="uint8"  start_byte="3" start_bit="4" bit_size="1"  codec="gbcsBitCo
        <field name="processing_option" type="uint8"  start_byte="3" start_bit="5" bit_size="1"  codec="gbcsBitCo
        <field name="service_class"     type="uint8"  start_byte="3" start_bit="6" bit_size="1"  codec="gbcsBitCo
        <field name="priority"          type="uint8"  start_byte="3" start_bit="7" bit_size="1"  codec="gbcsBitCo
  </structure>
    <structure name="date_time" codec="gbcsLengthValueCodec">
        <field name="length" type="uint8" codec="gbcsIntegerCodec" encoding="uint8" />
        <field name="value" type="datetime" codec="gbcsDateTimeCodec" />
    </structure>

    <structure name="access_request_body" codec="gbcsGenericStructureCodec">
        <structure name="access_request_specification" codec="gbcsGenericStructureCodec">
            <field name="number_of_requests" type="uint8" codec="gbcsIntegerCodec" encoding="uint8"/>
            <structure name="requests" codec="gbcsGenericStructureCodec">
                <structure name="supply_tamper_state_value" codec="gbcsGenericStructureCodec">
                    <field name="access_request_type" type="uint8" codec="gbcsIntegerCodec" encoding="uint8"/>
                    <structure name="attribute_descriptor" codec="gbcsGenericStructureCodec">
                        <field name="class_id" type="uint16" codec="gbcsIntegerCodec" encoding="uint16"/>
                        <field name="obis_code" type="string" codec="gbcsObisCodeCodec" max_length="24"/>
                        <field name="id" type="uint8" codec="gbcsIntegerCodec" encoding="uint8"/>
                    </structure>
                </structure>
            </structure>
        </structure>
        <structure name="access_request_list_of_data" codec="gbcsGenericStructureCodec">
            <field name="number_of_request_parameters" type="uint8" codec="gbcsIntegerCodec" encoding="uint8"/>
            <structure name="request_parameters" codec="gbcsGenericStructureCodec">
                <field name="supply_tamper_state_value" type="uint8" codec="dlmsBooleanCodec"/>
            </structure>
        </structure>
    </structure>

</structure>
```

**Figure 83 – Command Payload XML specification file overview.**

### Test Execution File

An XML file much in every way similar to the one described in Section 3.2. As an example, Figure 84 presents an overview of an assembled Test Properties file.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE executionconfiguration []>
<executionconfiguration>
  <testmanager master_node="svfmaster">
    <testspec class="CTestSpec" name="ATG-TEST-ECS81">
      <report>
        <item name="USE CASE">ECS81</item>
        <item name="TEST ID">ATG-TEST-ECS81</item>
        <item name="TEST PURPOSE">Test Use Case ECS81 &quot;Set Supply Tamper State on ESME&quot;, w
      </report>

      <testcase class="CTestCaseECS81" name="ATG-TC-ECS81">
        <report>
          <item name="TEST CASE ID">ATG-TC-ECS81</item>
          <item name="TEST CASE PURPOSE">The purpose of this test case is to validate Use Case ECS81
          <item name="TEST CASE ASSUMPTIONS AND CONSTRAINTS">N/A</item>
          <item name="AUTHOR">CSW</item>
          <item name="VERSION">01</item>
        </report>

        <resource_combiner class="CLinearCombinations" name="linearCombinator"/>

        <resourcepool class="CResourcePoolString" name="BusinessOriginatorName">
          <value>Supplier1</value>
        </resourcepool>

        <resourcepool class="CResourcePoolString" name="BusinessTargetName">
          <value>ESME1</value>
        </resourcepool>

        <resourcepool class="CResourcePoolString" name="SupplyTamperState.value">
          <value>TRUE</value>
        </resourcepool>

        <resourcepool class="CResourcePoolString" name="SupplyTamperState.result">
          <value>0</value>
        </resourcepool>

      </testcase>
    </testspec>

    <run times="1" />
  </testmanager>
</executionconfiguration>
```

**Figure 84 – Test Properties file overview.**

### 6.2.6 Use Case Configuration

On use-cases-messages.xml file from /opt/gfi/conf/usecases directory, add the following structure to add a custom command, response or alert. The following names (gcs01a and cs14) are merely an example for explanation purposes.

```xml
<message name="gcs01a_command" check_handler="KRP" codec="genericCodec">
    &gcs01aCommandPayload;
</message>
<message name="gcs01a_response" check_handler="KRP" codec="genericCodec">
    &gcs01aResponsePayload;
</message>
```

**Figure 85 – Custom Command and Response configuration.**

```
<message name="cs14_alert" check_handler="KRP" codec="genericCodec">
&cs14AlertPayload;
</message>
```

Figure 86 – Custom Alert configuration.

### 6.2.7 Custom Message Code

Add the new message code on gbcsconstants.h file from /opt/gfi/src/base directory.

```
//Message codes: Gas use-cases
#define MESSAGE_CODE_CS01B  0x0097
#define MESSAGE_CODE_CS03B  0x000E
#define MESSAGE_CODE_CS10A  0x0014
#define MESSAGE_CODE_CS10B  0x00A1
#define MESSAGE_CODE_CS11   0x0015
#define MESSAGE_CODE_GCS01A 0x006B
#define MESSAGE_CODE_GCS01B 0x00A3
```

Figure 87 – Example of a Message Code definition.

### 6.2.8 Match Message Code and Message Type

On cgbcsutils.cpp file from /opt/gfi/src/base directory add the corresponding code using the corresponding message code. This will match the message code and message type (Command, Response or Alert) to a specific message structure.

```
mapTemp[make_pair(make_pair(MESSAGE_CODE_GCS01A, CRA_FLAG_COMMAND), 0)] = GCS01A_COMMAND::MSG_ID;
mapTemp[make_pair(make_pair(MESSAGE_CODE_GCS01A, CRA_FLAG_RESPONSE), 0)] = GCS01A_RESPONSE::MSG_ID;
```

Figure 88 – Example of a Custom Use Case matching a Command and a Response.

```
mapTemp[make_pair(make_pair(MESSAGE_CODE_CS14, CRA_FLAG_ALERT), 0)] = CS14_ALERT::MSG_ID;
```

Figure 89 - Example of a Custom Use Case matching an Alert.

Please note that if the custom Use Case is critical, it is required to add the message code on the method IsCriticalMessageCode().

```
bool CGBCSUtils::IsCriticalMessageCode(uint16_t un_message_code)
{
    return (un_message_code == MESSAGE_CODE_GCS01A ||
            un_message_code == MESSAGE_CODE_CS02B_ROOT_BY_SUPPLIER ||
```

Figure 90 – Message Code added to IsCriticalMessageCode() method.

## 6.3 Use Case Coding

The Use Case Coding is the second phase in the process of creating a new Use Case. Knowledge of object-oriented programming and C++ is required to successfully execute this phase. The framework is implemented in the C++ language using Eclipse Luna and build over the base libraries of CSW VSIS™.

During this phase, a set of C++ classes must be produced and coded in order to add the new Use Case to be available for the GFI testing tool Use Case Library. Typically five C++ classes are produced in this phase:

- Use Case Base class;
- Use Case Command class;
- Use Case Response class;
- Use Case Alert class (if applicable);
- Test Case class.

The Emulators (ESME, GSME and CHF) classes must be updated to support the new Use Case. In the emulators, the User typically has to code the Command handler, Response handler and Alert handler (if applicable) methods. After coding all these classes and methods, the Use Case is ready to be compiled and integrated in the GFI testing tool Library. Figure 91 presents an overview of the Coding phase. The following Sections will explain all the sub-phases.

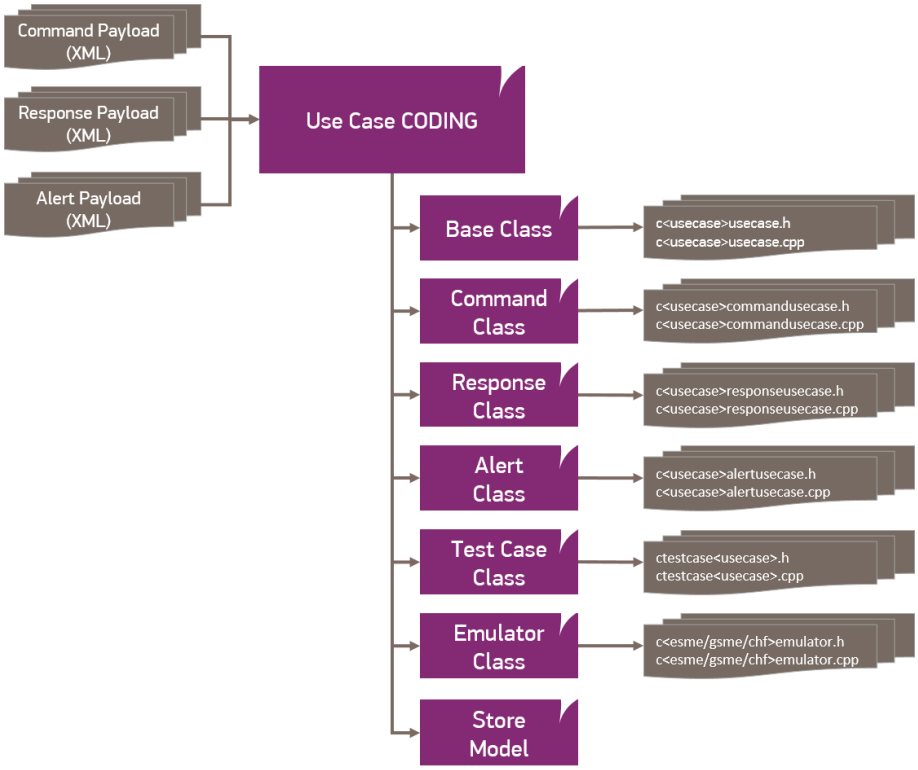


**Figure 91 – Use Case Coding phase overview.**

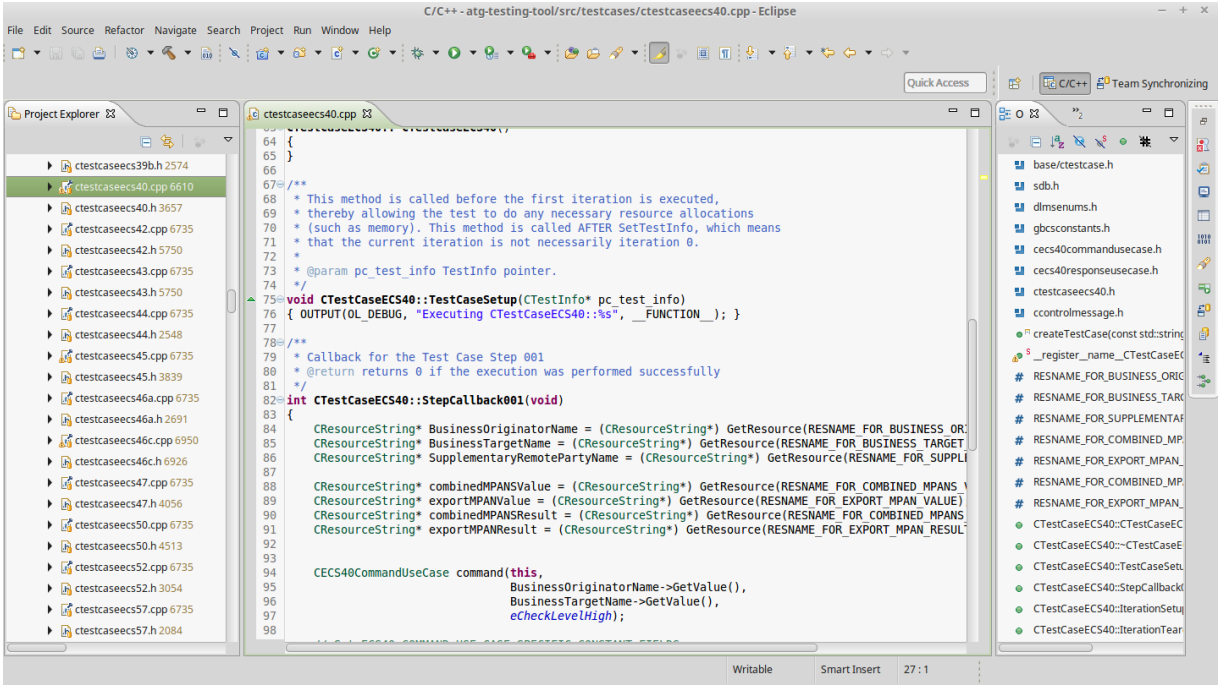


**Figure 92 – Eclipse Luna IDE overview.**

### 6.3.1 Base Class Files

These files are located in the project folder under /src/usecases/<protocol>/ (where <protocol> is whichever protocol the Use Case relates to: DLMS, ZigBee, ASN1) and must be created manually from the provided templates. Also, the existing Use Case specifications and code may be used as example to create these new code files. They are:

c<usecase>usecase.h

c<usecase>usecase.cpp

These files contain the Use Case class which will be inherited by the Use Case message classes. They also include some common methods and definitions.

### 6.3.2 Command Class Files

These files are located in the project folder under /src/usecases/<protocol>/ (where <protocol> is whichever protocol the Use Case relates to: DLMS, ZigBee, ASN1) and must be created manually from the provided templates. Also, the existing Use Case specifications and code may be used as example to create these new code files. They are:

c<usecase>commandusecase.h

c<usecase>commandusecase.cpp

These files contain the command class, command specific definitions and methods for setting constant and variable input parameters.

### 6.3.3 Response Class Files

These files are located in the project folder under /src/usecases/<protocol>/ (where <protocol> is whichever protocol the Use Case relates to: DLMS, ZigBee, ASN1) and must be created manually from the provided templates. Also, the existing Use Case specifications and code may be used as example to create these new code files. They are:

c<usecase>responseusecase.h

c<usecase>responseusecase.cpp

These files contain the response class, response specific definitions and methods for checking output parameters. In the case of emulated metering equipment being used, it also includes methods to setup the response message and to invoke the emulator functions.

### 6.3.4 Alert Class Files

These files are located in the project folder under /src/usecases/<protocol>/ (where <protocol> is whichever protocol the Use Case relates to: DLMS, ZigBee, ASN1) and must be created manually from the provided templates. Also, the existing Use Case specifications and code may be used as example to create these new code files. They are:

c<usecase>alertusecase.h

c<usecase>alertusecase.cpp

These files contain the alert class, alert specific definitions and methods for checking output parameters. In the case of emulated metering equipment being used, it also includes methods to setup the alert message and to invoke the emulator functions.

### 6.3.5 Test Case Class files

These files are located in the project folder under /src/testcases/ and must be created manually from the provided templates. Also, the existing Use Case specifications and code may be used as example to create these new code files. These files are:

ctestcase<usecase>.h

ctestcase<usecase>.c

Contain the test case class and the test case specific definitions. Also contains the action sequence described in the 'Test Cases' tab.

### 6.3.6 Emulator Class

When using emulated metering devices the required emulator actions should be coded in files used specifically for this purpose. These files are located in the project folder under /src/emulators/ and there are a number of files to differentiate the communication protocols implemented in the framework. By general principle each set of actions will be coded in a device related file (ESME, GSME or CHF). For example, ESME related Use Case emulation will be coded in a different file than GSME related Use Case emulation.

The emulator's command, response and alert handlers are invoked with a message code, it will then switch to the Use Case handler concerning that message. The basic emulator function is bound to invoke a routine to produce output information but further logic may be included to reproduce the device behaviour.

### 6.3.7 Store Model

The Store Model is necessary when using emulated devices, as it contains information regarding the equipment's attributes, trust anchor cells and device private keys. The files located at /src/storemodel/ hold the methods for the emulator to interact with the stored information in the scenario configuration file.

### 6.3.8 Compilation

After completing the Use Case's coding, the source code must be compiled in order to produce a new executable testing tool. To do so, the User has two options: a) manually compile the tool using the command-line; b) use Eclipse CDT.

#### Using the Command-line

To compile the tool using the command-line, the User just have to follow the following procedure:

Open a terminal;

Execute the command "cd /opt/gfi/scr";

Execute the command "make".

If everything goes as expected the compilation of the tool should start, similar to the presented in Figure 93, and in the current working directory a new "gfi-testing-tool" executable will be created.

```
/opt/gfi/src $ make
g++  -I. -I /opt/gfi/src -I /opt/gfi/src/sdb -I /opt/gfi/src/base -I /o
tors -I /opt/gfi/src/emulators/utils -I /opt/gfi/src/cryptography -I /o
model/zigbee -I /opt/gfi/src/codecs/utils -I /opt/gfi/src/storemodel -I
gfi/src/usecases/zigbee -I /opt/gfi/src/usecases/asn1 -I /opt/gfi/src/u
/parsec -I/usr/include/json-c          -fpermissive -std=c++11 -DOS_POSI
ggc-min-expand=20 -DGFI_VERSION_STRING="GFI Custom Build [gfi-custom-bu
g++ -o gfi-testing-tool ./sdb/sdb.o ./emulators/cesmeemulator.o ./emula
.o ./gfi-main.o ./testcases/ctestcaseecs26d.o ./testcases/ctestcaseecs2
ses/ctestcaseecs34.o ./testcases/ctestcaseecs20d.o ./testcases/ctestcas
tcases/ctestcaseecs44.o ./testcases/ctestcasegcs21d.o ./testcases/ctest
./testcases/ctestcaseecs05.o ./testcases/ctestcasegcs21f.o ./testcases/
o ./testcases/ctestcasegcs21a.o ./testcases/ctestcaseecs27.o ./testcase
19.o ./testcases/ctestcasecs11.o ./testcases/ctestcaseecs18a.o ./testca
cs05.o ./testcases/ctestcasegcs39.o ./testcases/ctestcaseccs02.o ./test
ecs04.o ./testcases/ctestcasegcs21j.o ./testcases/ctestcasegcs01b.o ./t
caseecs28a.o ./testcases/ctestcasecs01b.o ./testcases/ctestcasegcs40a.o
testcasecs02a.o ./testcases/ctestcasezigbeegnca.o ./testcases/ctestcase
stcases/ctestcaseecs15a.o ./testcases/ctestcaseecs17e.o ./testcases/cte
o ./testcases/ctestcaseecs26a.o ./testcases/ctestcasegcs13b.o ./testcas
s38.o ./testcases/ctestcasecs07.o ./testcases/ctestcasedlms_fdra.o ./te
```

Figure 93 – GFI Tool compilation using the command-line.

Getting the version of the new testing tool will print something similar to the presented in Figure 94. As one can see, this is a custom build of the GFI Testing Tool, so invoking the get version command-line command will inform the User about that.

```
/opt/gfi/src $ ./gfi-testing-tool --version
GFI Custom Build [gfi-custom-build] (Sep 11 2015 17:03:38)
Copyright (c) Smart DCC Ltd.
```

Figure 94 – Custom GFI Testing Tool version.

Because no version number was specified by the User, the text "gfi-custom-build" is presented between "[" and "]". If desired, a version number can be specified using the environment variable "VERSION". In order to specify a version number to the Testing Tool, the User can execute the command "make" as presented in Figure 95.

```
/opt/gfi/src $ make VERSION="MY-VERSION"
g++  -I. -I /opt/gfi/src -I /opt/gfi/src/sdb -I /opt/gfi/src/base -I /opt/gfi/src
tors -I /opt/gfi/src/emulators/utils -I /opt/gfi/src/cryptography -I /opt/gfi/src
model/zigbee -I /opt/gfi/src/codecs/utils -I /opt/gfi/src/storemodel -I /opt/gfi/
gfi/src/usecases/zigbee -I /opt/gfi/src/usecases/asn1 -I /opt/gfi/src/usecases -I
/parsec -I/usr/include/json-c         -fpermissive -std=c++11 -DOS_POSIX -DOS_LIN
ggc-min-expand=20 -DGFI_VERSION_STRING="GFI Custom Build [MY-VERSION]" -c gfi-mai
g++ -o gfi-testing-tool ./sdb/sdb.o ./emulators/cesmeemulator.o ./emulators/cchfe
.o ./gfi-main.o ./testcases/ctestcaseecs26d.o ./testcases/ctestcaseecs28b.o ./tes
ses/ctestcaseecs34.o ./testcases/ctestcaseecs20d.o ./testcases/ctestcasegcs13a.o
tcases/ctestcaseecs44.o ./testcases/ctestcasegcs21d.o ./testcases/ctestcaseecs22a
```

Figure 95 – GFI Tool compilation specifying a version.

### Using Eclipse CDT

If using Eclipse CDT to compile the testing tool, the User just have to click in the build button, as presented in Figure 96.

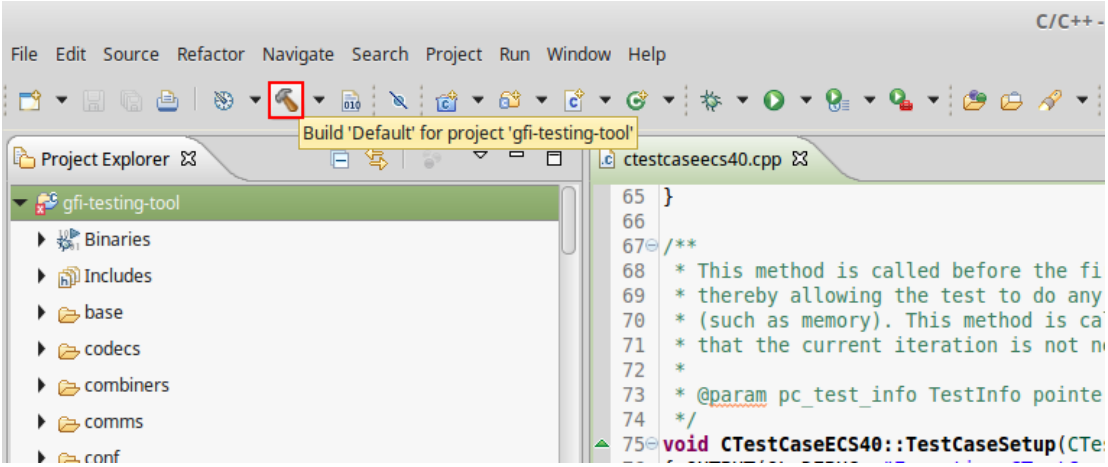


Figure 96 – GFI Tool compilation using Eclipse CDT.

As with the command-line, if everything goes as expected the compilation of the tool should start and in the current working directory a new “gfi-testing-tool” executable will be created.

In order to specify the environment variable VERSION while using Eclipse CDT, the User should access to the menu “Project➔Properties”, then “C/C++ Build➔Environment” and add the variable VERSION, as presented in Figure 97, with the desired value (“2.0-MY-VERSION” in the provided example).

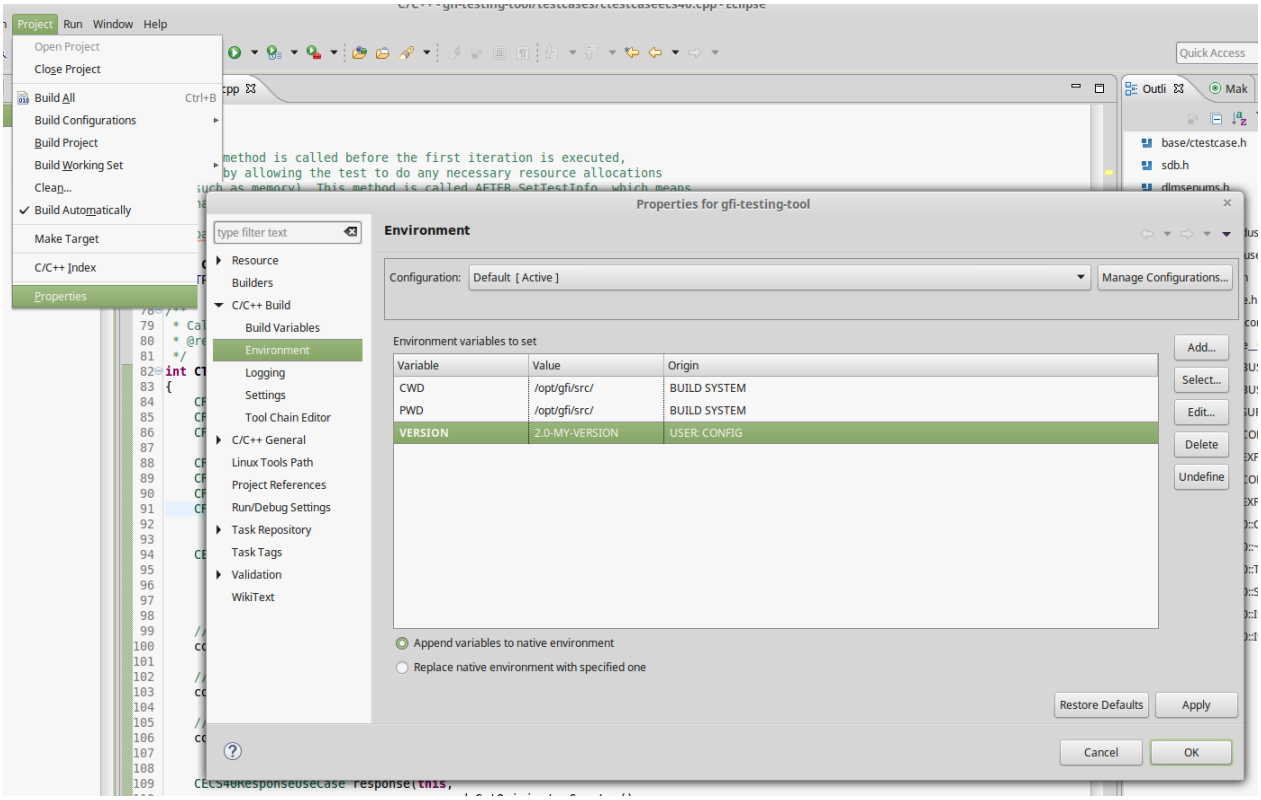


Figure 97 – Setting environment variables in Eclipse CDT.

### 6.3.9 Installation

After a successful compilation, the User can install the current GFI testing tool, freezing a specific version into a specific folder. The installation process will copy all the necessary files into the folder "/opt/gfi/custom/<gfi-version>-<datetime>", where <gfi-version> will be equal to the VERSION environment variable's value and <datetime> will be the installation date and time. Again, the User have two options: a) use the command-line; b) use the Eclipse CDT.

#### Using the Command-line

To install the tool using the command-line, the User just have to follow the following procedure:

Open a terminal;

Execute the command "cd /opt/gfi/scr";

Execute the command "make VERSION="<gfi-version>" install".

If everything goes as expected the installation of the tool should start and in the directory "/opt/gfi/custom/" a new folder with the tool installation will be created, similar to the presented in Figure 98.

Figure 98 –GFI Testing Tool after installation.

### Using Eclipse CDT

To install the tool using Eclipse, the User should access to the menu “Project→Make Target→Build…”, select the target “Install” and click the button “Build”, as presented in Figure 99.

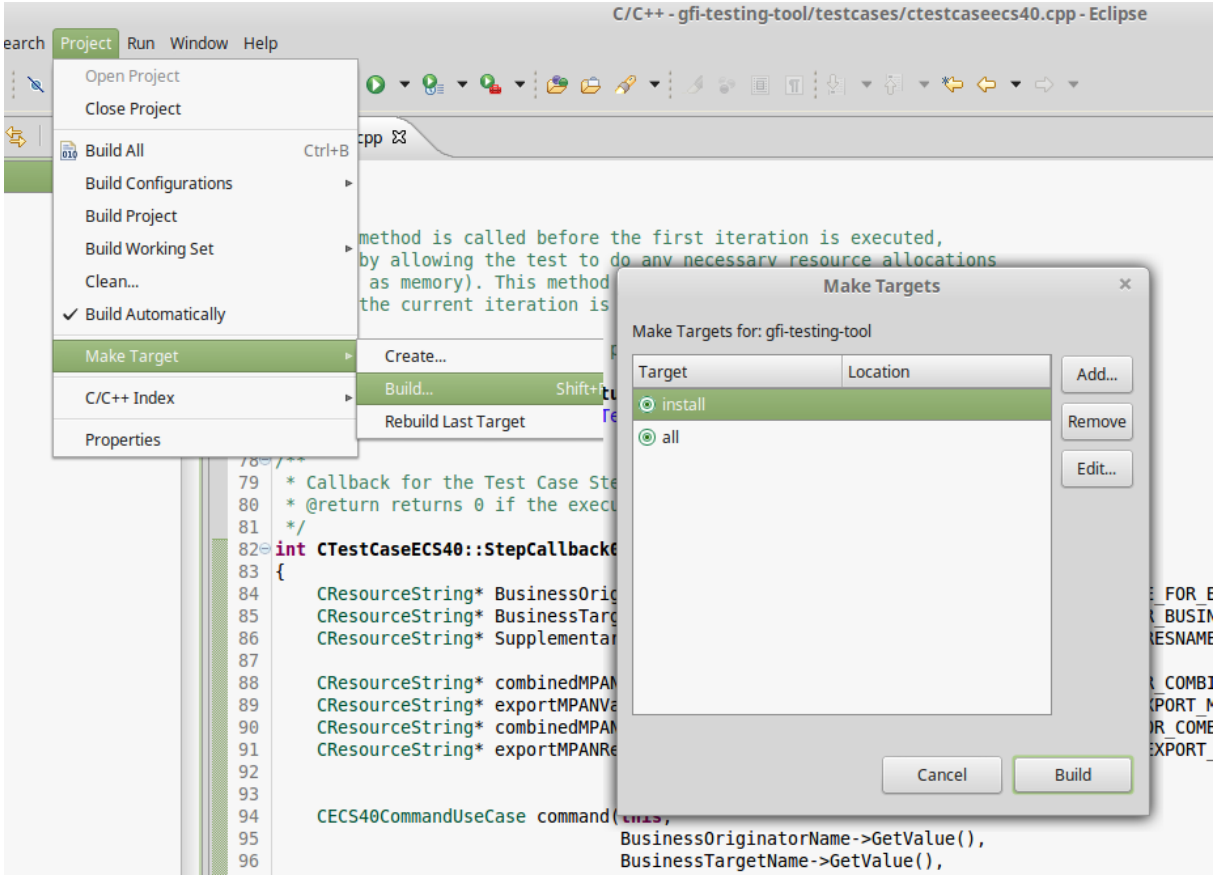


Figure 99 – GFI Testing tool installation using Eclipse CDT.

## 6.4 Use Case Execution

The Use Case Execution is the third phase in the process of creating a new Use Case. This is probably the most simple one. The GFI testing tool is executed with the Execution XML file produced in the first phase as one of the inputs and an HTML report is produced by the tool. See Section 5 for further details.

Figure 100 presents an overview of the Execution phase.

During the Execution phase, errors and crashes may arise if the User is adding a new Use Case. Is responsibility of the User to investigate and correct all the errors that may have been introduced during the Coding phase. Again, knowledge of C++ is required to solve the eventual errors and warnings.

Figure 100 – Use Case Execution phase overview.

### 6.4.1 Execution with GUI

In the event the User wishes to execute the custom GFI Tool compiled in the previous phase with the GFI GUI, the path to the executable file should be configured. To do so, in the GUI the User should access to the menu “Edit→Preferences” and configure the path to the desired executable Testing Tool file. By default the GUI uses the Tool available at “/opt/gfi/”.

## 6.5 Use Case Validation

The last phase is the Use Case Validation. The User must check if the Use Case execution succeeded and if all the payloads, Command, Response and Alert, agree with the GBCS, DLMS and ZigBee specification. In the event of all payloads are according to the specifications, the Use Case creation is done, it is now available in the GFI Test Library and ready to be integrated in a Test. If a mismatch is found on the payloads, the User should make the necessary corrections/modifications in the specification and the process starts all over again from the beginning. Figure 101 presents an overview of the Validation phase.

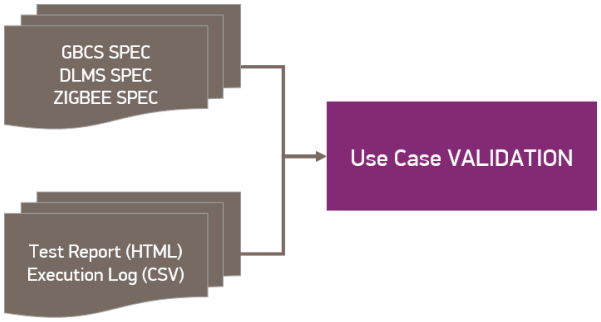


**Figure 101 – Use Case Validation phase overview.**

# 7 Adding New Test Cases to the GUI

This section is intended for Advanced Users. The topics presented in this section will have direct impact in the GUI’s operation. Experience in understanding and editing XML files is required. Users should keep this in mind.

The previous section provided a detailed explanation about the process of extending the existing test case database by creating new test-cases or altering the existing ones - from the initial specification, message/test files generation, coding and execution.

It was mentioned briefly the GUI could be pointed to a new GFI distribution by changing the path to the gfi-testing-tool binary in the GUI’s preferences dialog window. However, regardless of the GFI executable configured to run, the GUI will always provide the same set of test cases by default (ie the ones shipped with the default distribution). During the GUI’s test construction phase, the original test case database will always be available. It is, however also possible to make newly created test cases visible to the GUI application, and so utilize them in a new test.

## 7.1 Test Case Database File


```xml
<?xml version="1.0" encoding="UTF-8"?>
<TestCases>
  <TestCase id="ATG-TC-CS01a-ESME">
    <Purpose>The purpose of this test case is to validate Use Case CS01a "Apply Prepayment Top Up"</Purpose>
    <Class>CTestCaseCS01a</Class>
    <UseCase ContainsSensitiveData="No" DateTimeStampInResponse="No"
    FutureDatedInvocation="No" KeyAgreementCertificate="No" ProtectionAgainstReplay="Yes"
    SupplementaryOriginatorCounter="No" SupplementaryRemoteParty="No" id="CS01a">
      <Name>Apply Prepayment Top Up to an ESME</Name>
      <Description>This Use Case enables a UTRN to be applied to an ESME to top up the meter balance. See also section 14.</Description>
      <Protocol>DLMS</Protocol>
      <MessageCode>0x0007</MessageCode>
      <RemotePartyRoles>
        <RemotePartyRole>Supplier</RemotePartyRole>
      </RemotePartyRoles>
      <MessageType>Command and Response</MessageType>
      <MessageTypeCategory>Variant Command and Response</MessageTypeCategory>
      <ReadOrUpdate>Update</ReadOrUpdate>
      <TargetDevices>
        <TargetDevice>ESME</TargetDevice>
      </TargetDevices>
    </UseCase>
    <AssumptionsAndConstraints>N/A</AssumptionsAndConstraints>
    <Filter>ESME</Filter>
    <Parameters>
      <Parameter name="Control.Error.Code" type="Input" resourceType="CResourcePoolString" sourceType="value" control="Error">
        <Value>0</Value>
        <Value>0</Value>
      </Parameter>
      <Parameter name="BusinessOriginatorName" type="Input" resourceType="CResourcePoolString" sourceType="value">
        <Value>Supplier1</Value>
        <Value>Supplier1</Value>
      </Parameter>
      <Parameter name="BusinessTargetName" type="Input" resourceType="CResourcePoolString" sourceType="value">
        <Value>ESME1</Value>
        <Value>ESME1</Value>
      </Parameter>
      <Parameter name="PrepaymentCredit.creditToken" type="Input" resourceType="CResourcePoolString" sourceType="value">
        <Value>74301825837596672005</Value>
        <Value />
      </Parameter>
      <Parameter name="PTUTValueClass" type="Input" resourceType="CResourcePoolString" sourceType="value">
        <Value>1</Value>
        <Value>0</Value>
      </Parameter>
      <Parameter name="PTUTValue" type="Input" resourceType="CResourcePoolString" sourceType="value">
        <Value>10</Value>
        <Value>1234</Value>
      </Parameter>
      <Parameter name="PrepaymentCredit.creditToken.result" type="Output" resourceType="CResourcePoolString" sourceType="value">
        <Value>0</Value>
        <Value>0</Value>
      </Parameter>
    </Parameters>
  </TestCase>
```


**Figure 102 – The file testcasedata.xml.**

As was mentioned in the previous section, the GUI application is shipped with a default test case database file. This file is testcasedata.xml, and is copied to a hidden directory in the user's home directory (.gui) every time the application is started. A snapshot of this file is shown in Figure 102.

The information is collected from several sources including the XML specification files, but also from the GBCS Mapping table. Some additional tags control what and how the GUI will display the information. The tag <TestCases> is the root element, and each test case is a <TestCase> element, representing a test case, with an id attribute to identify it. The <Class> element represents the class implementing the test case and must to be unique.

The <TestCase> element has a child element containing the respective use case information: <UseCase>. This information is displayed when the Add New Test Case dialog window is opened. The contents of the <Name> element will be used to identify the test case.

Of special importance is the <Filter> element, as it defines how the test case will be visible. Possible values are CHF, GPF, ESME, GSME, PPMID and HCALCS. This will make the test visible only when it is for the device type specified.

The <Parameters> section defines the test case parameters, as well as their default values. It closely resembles the contents of a test execution file (properties file). These are the default iteration values used whenever a test is created using the GUI wizard. Most of the <Parameter> element attributes will map directly to their equivalent in the test execution file. However, the attribute longString defines the type of component used in the GUI's visual editor to edit the value of the parameter: if set to True, the component will be a multiline text box; otherwise it will be a single-line text box.

## 7.2 Manufacturer Test Cases File

To make new test cases visible in the GUI, a new file called manufacturer-test-cases.xml must be present in the .gui directory, and its format must be the exactly the same as the testcasedata.xml file. The testcasedata.xml file may serve as a template for its creation, but the user is advised to make sure no test case names or classes are duplicated, and the structure is strictly observed, otherwise the file contents may be ignored. Next, an example will be presented to illustrate the concepts just described.

The default database is shipped with two test cases with variations for ESME and GSME: CS03 and CS04. For CS03, the names are CS03-ESME and CS03-GSME respectively. Their implementation is exactly the same, the only difference being the parameter values. The test cases have different names and class names but they extend the same class: CTestCaseCS03. Having this in mind, and solely for the sake of illustrating the concept, the original test case could be included adding it to the manufacturer-test-cases.xml file, provided the names for it and its class are unique. Figure 103 shows how this could be done.

```
Terminal
<?xml version="1.0" ?>
<TestCases>
    <TestCase id="ATG-TC-CS03-ORIGINAL">
    <Purpose>The purpose of this test case is to validate Use Case CS03 &quot;Join Device&quot; by validating the construction of
 the command used to add the ability of the target Device to communicate with other given Device in the HAN and its respective re
sponses.</Purpose>
    <Class>CTestCaseCS03</Class>
    <UseCase ContainsSensitiveData="No" DateTimeStampInResponse="No" FutureDatedInvocation="No" KeyAgreementCertificate="No" Prot
ectionAgainstReplay="Yes" SupplementaryOriginatorCounter="No" SupplementaryRemoteParty="Yes" id="CS03">
      <Name>Method A Join (Meter)/Method B Join/Method C Join</Name>
      <Description>See section 13.7.</Description>
      <Protocol>ASN.1</Protocol>
      <MessageCode>0x000D - CS03A1 - Method A Join;
0x000E - CS03B - Method B Join.</MessageCode>
      <RemotePartyRoles>
        <RemotePartyRole>Supplier</RemotePartyRole>
        <RemotePartyRole>Access Control Broker (only on CS03B - Method B Join)</RemotePartyRole>
      </RemotePartyRoles>
      <MessageType>Command and Response</MessageType>
      <MessageTypeCategory>SME.C.C (CS03A1 - Method A Join / CS03A2 - Method A Join - non Meter / CS03C - Method C Join);
SME.C.NC (CS03B - Method B Join).</MessageTypeCategory>
      <ReadOrUpdate>Update</ReadOrUpdate>
      <TargetDevices>
        <TargetDevice>ESME</TargetDevice>
      </TargetDevices>
    </UseCase>
    <AssumptionsAndConstraints>Issue 5000/IRP 269 : CS03A2 should be SME.C.C
Issue 5060: CS03C shall not have the &quot;Supplementary Remote Party Data required?&quot; marked with &quot;yes&quot; in mapping
 table.</AssumptionsAndConstraints>
    <Filter>ELECTRIC</Filter>
    <Parameters>
      <Parameter name="BusinessOriginatorName" resourceType="CResourcePoolString" sourceType="value" type="Input">
        <Value>Supplier1</Value>
        <Value>ACB</Value>
      </Parameter>
```

**Figure 103 – manufacturer-test-cases.xml.**

The TestCase element is a copy of the ESME variation; However the test case id is ATC-TC-CS03-ORIGINAL and the class name is CTestCaseCS03. These must be unique values. As the filter value is ELECTRIC, this test case will be added to the list of electric test cases (Figure 104).

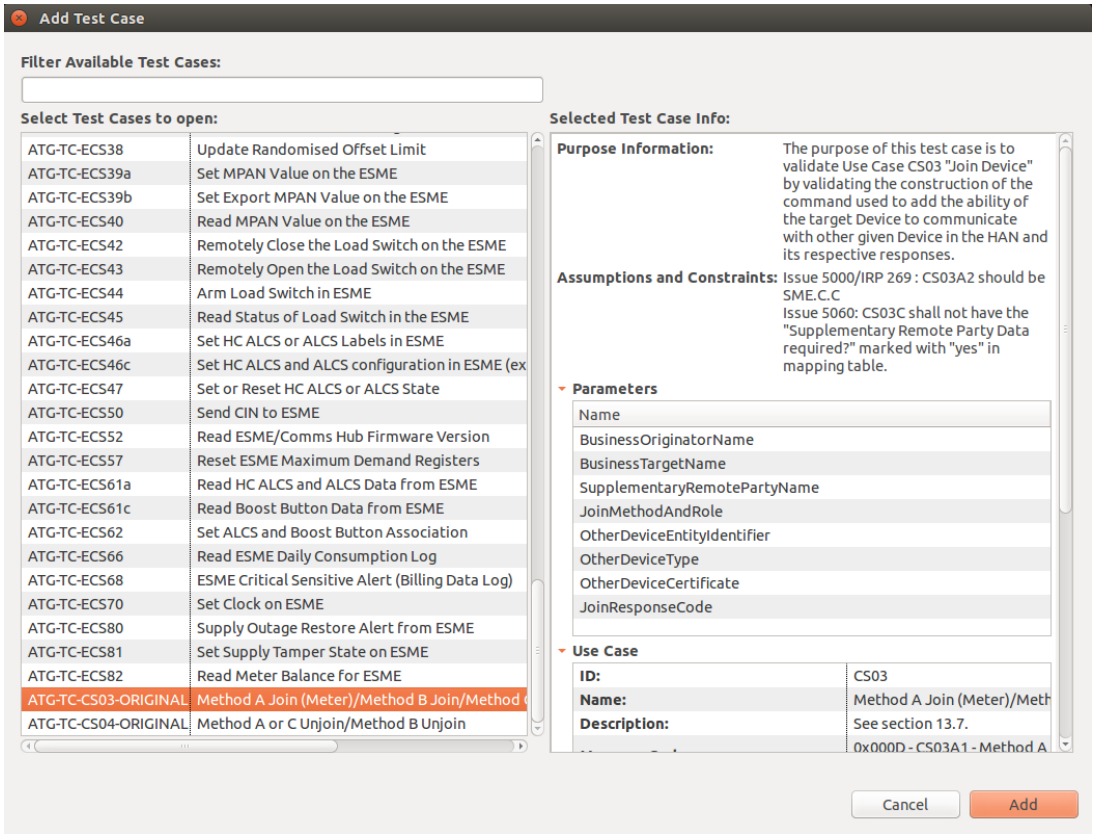


Figure 104 – Updated list of Test Cases.

# 8 Business Scenarios

The GFI tool is used to control and test SMETS 2 devices and its use is limited by the user's knowledge of GBCS. In order to allow the user to focus on the intended section of GBCS, starting on GFI version 2.1.6, a set of pre-prepared business scenarios (workspace and manuals) was made available, allowing a user to easily set up a device and to test that specific scenario.

The business scenarios (workspace and manuals) are located in **/opt/gfi/doc/business-scenarios/** in a separate folder for each GBCS version it is aligned with.

For each GBCS version, there is a global manual "business-scenarios-manual.pdf" that gives an overview on the organization and usage of these scenarios.

The user can also access the business scenarios folder using the shortcut from the desktop.

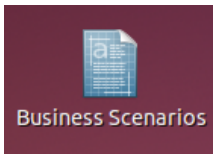


Figure 105 - Business Scenarios Desktop shortcut

# 9 OTA Functionality

## 9.1 OTA Tool

The GUI has a functionality to create and verify OTA Upgrade Images which can be accessed by using the option “OTA Tool” inside of “Utilities” (Figure 93).

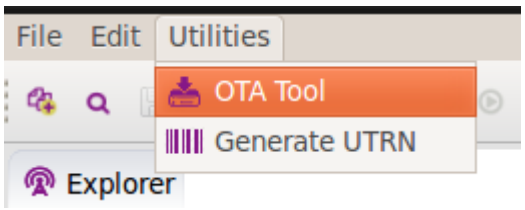


**Figure 106 – Invoking the ‘OTA Tool’ functionality through the Utilities menu**

In the OTA Tool there are two main options: Build OTA Upgrade Image and Verify Images Info (Figure 94).

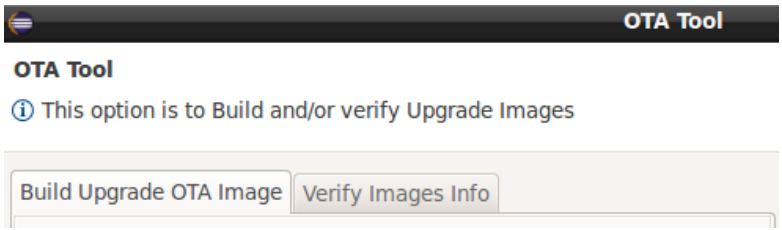


**Figure 107 – Build and Verify tabs**

The Build tab is used to build an OTA Upgrade Image and the Verify tab to verify if the OTA Upgrade Image or if the input files for the OTA Upgrade Image build are valid.

In both tabs the mandatory fields are marked with a ‘*’ and, depending on the Build or Verify types selected, the other options will change to avoid mistakes using the tool.

The form errors will be shown at the Dialog Header at the info section (Figure 95).

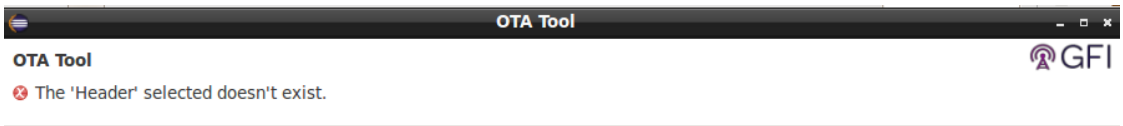


**Figure 108 – Form errors**

For both cases (build and verify) the log of the operation will be shown on the right side of the window. In this section the form errors and the process debug information are displayed (Figure 96).

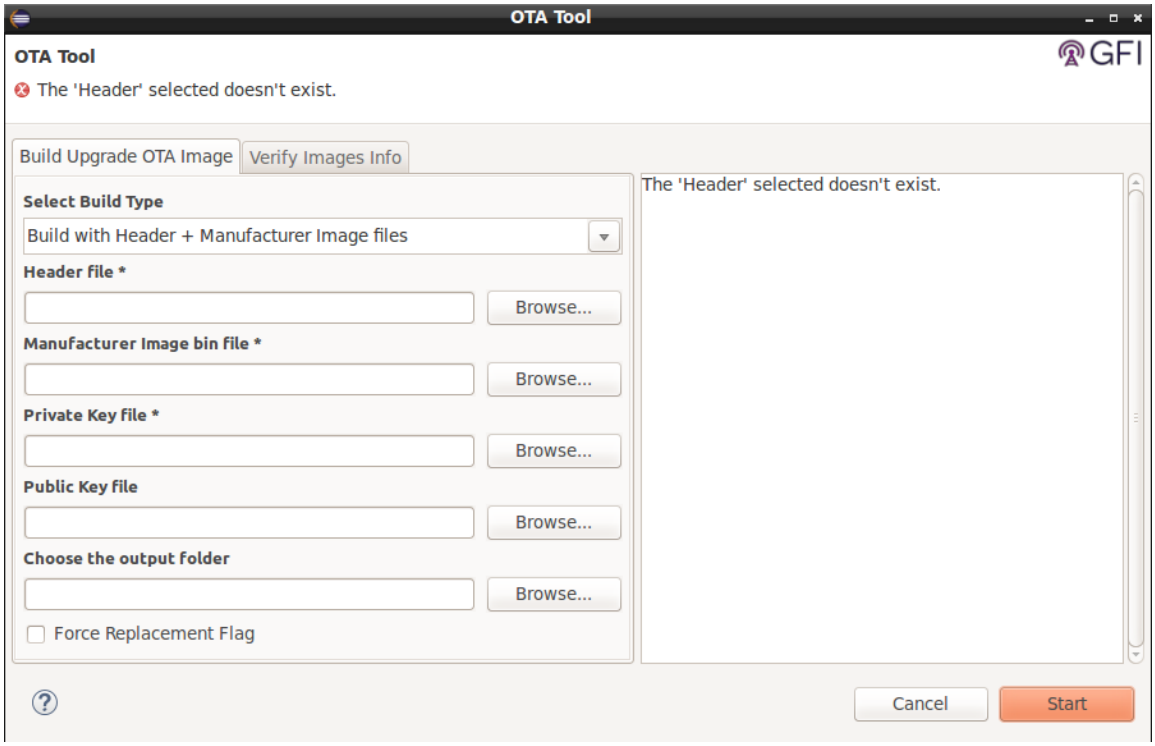


**Figure 109 – Log section**

It is also possible to use the “OTA Tool” from the command line. The “OTA Tool” executable is available on the folder “/opt/gfi/bin” and when running with option “--help”, the list of parameters is displayed.

### 9.1.1 Build OTA Upgrade Image

In the Build tab, the Manufacturer Image can be made available in two different ways: two separated files (Header + Manufacturer Image) or in a Single file containing both contents. To select the correct type use the option “**Select Build Type**”.

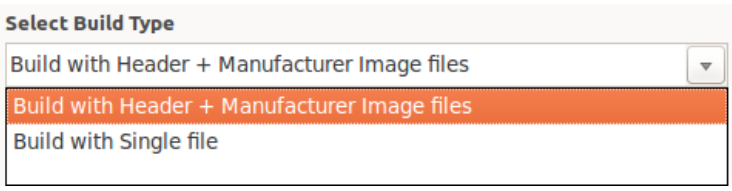


**Figure 110 – Choose Build type**

To build an Upgrade Image the Private Key (to sign the output file) and Header and Manufacturer Image (in separated files or in a single file) files are mandatory.

The Public Key file is used only to verify if the Upgrade Image has been correctly signed and the Output folder is used to allow the user to specify a folder where the Upgrade Image will be saved. If the

folder is not specified, the Upgrade Image will be created on the user Desktop and will be named as **upgradeImage.bin**.

When build process finishes, the Upgrade Image will always be validated using the Header information and if the Public Key has been provided, the Signature will also be validated.

From the command line, to build an OTA image using a file with the header and a file with the manufacturer image, use the following command:

```
/opt/gfi/bin/otaTool -b -H header.bin -M manufacturer-image.bin -P
/opt/gfi/conf/smki/org/ZAZ1/key/Z1-supplier-ds.key
```

To build an OTA image using a file containing the header concatenated with the manufacturer image, the following command must be used:

```
/opt/gfi/bin/otaTool -b -S header-img.bin -P /opt/gfi/conf/smki/org/ZAZ1/key/Z1-
supplier-ds.key
```

### 9.1.2 Verify OTA Upgrade Image

To use the **Verify Upgrade OTA Image Data** option, the only mandatory information is the Upgrade Image itself but if the user also provides the Public Key, then the OTA Tool will also check the signature of the Upgrade Image. Otherwise, only the header will be verified.

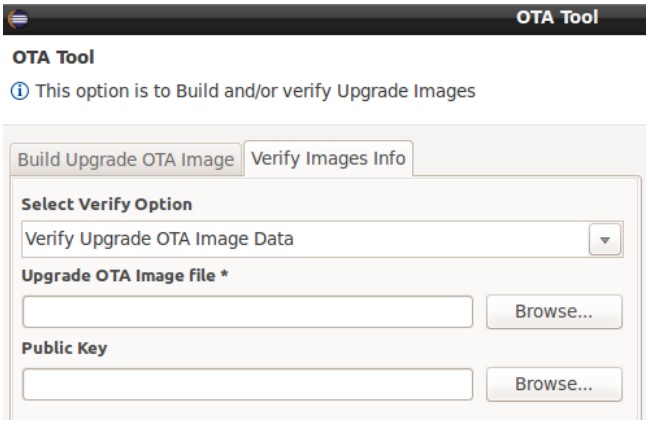


**Figure 111 – Verify Upgrade Image**

To verify the OTA upgrade image using the command line, use the following command:

```
/opt/gfi/bin/otaTool -v -U upgradeImage.bin -p /opt/gfi/conf/smki/org/ZAZ1/cert/Z1-
supplier-ds.pem
```

### 9.1.3 Verify Manufacturer Image construction

To verify if the Header file or/and the Signature are valid for the Manufacturer Image, use the 'Verify Manufacturer Image construction' option.

If the purpose of the verification is to check if the Header is valid, the user needs to input the Header file. If the user needs to verify the Signature, it is required to provide the Signature file and the Public Key. For both cases, the Manufacturer Image File is mandatory.

These two verifications can be run at once by providing all the information for both verifications.

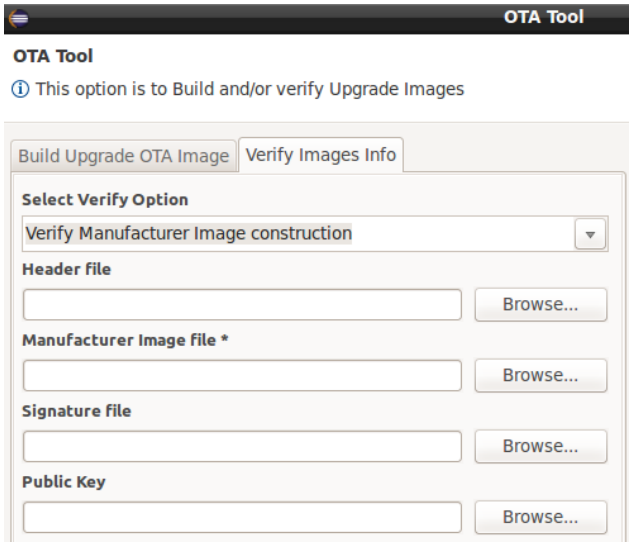


**Figure 112 – Verify Manufacturer Image**

## 9.2 Send Image to Devices

The OTA upgrade images to send to the devices need to be available on the Communications Hub storage folder. If the user uses the default location to store the images, the path is /opt/gfi/ota-files. In the event that the user chooses a different folder (in scenario file), the images should be stored in this location. To change the OTA storage folder, please refer to 4.1.4.

The images in the folder will appear in the Communications Hub Log, when the device is started.

```
14:43:30.591 |  STACK |  4 | GPF: hiding endpoint status: 0x00
Found OTA file 'gsme_GFI_FFFFFFFF.ota'
  Manufacturer ID: 0x121D
  Image Type ID:   0x0001
  Version:         0xFFFFFFFF
  Header String:   I am a GFI GSME[] firmware image
Found OTA file 'esme_GFI_FFFFFFFF.ota'
  Manufacturer ID: 0x121D
  Image Type ID:   0x0000
  Version:         0xFFFFFFFF
  Header String:   I am a GFI ESME[] firmware image
14:43:30.621 |  STACK |  4 | Found 2 files
```

**Figure 113 - Image files in Communications Hub Storage**

When a new image is added to the folder after the Communications Hub is running, the user needs to update the files in the folder choosing the command '**ota check-images**' in the Communications Hub CLI.

### 9.2.1 Send Image notify to devices by node Id

To notify the devices for the existence of a new OTA upgrade image in the Communications Hub, the user needs to choose the '**ota send-image-notification unicast**' command in the Communications hub followed by the device node to send the notification for new image. If the device is identified as a GSME, only the respective FNF is actioned. If the device is identified as a non-GSME, an image-notify command will be sent to the target.

### 9.2.2 Send Image notify to all devices by broadcast

To notify all devices, the command '**ota send-image-notification broadcast**' must be used to send the new image notification. The Communications hub sends the image-notify command to broadcast and also activates the respective FNF for GSME.

# 10 Glossary

Table 19 presents the list of definitions used throughout this document.

| Name | Description |
|---|---|
| Applicable Document | A document is considered applicable if it complements this document. All its content is directly applied as if it was stated as an annex of this document. |
| Reference Document | A document is considered a reference if it is referred but not applicable to this document. Reference documents are mainly used to provide further reading. |

| Name | Description |
|---|---|
| Test | A set of Test Cases (ranging from one to N) that are grouped together with the purpose of being executed in one run. |
| Test Case | A Use Case as defined by SMETS. |
| VSIS™ | A CSW Critical Systems Validation Platform |

**Table 19 – Definitions.**

Table 20 presents the list of acronyms used throughout this document.

| Acronym | Description |
|---|---|
| ACB | Access Control Broker |
| AD | Applicable Document |
| ASN.1 | Abstract Syntax Notation One |
| ATG | Automated Test of GBCS |
| CBKE | Certificate-Based Key Exchange |
| CHF | Communications Hub Function |
| CSW | Critical Software, S.A. |
| DUT | Device Under Test |
| ESME | Electricity Smart Metering Equipment |
| EUI-64 | The IEEE-defined 64-bit Extended Unique Identifier |
| GBCS | Great Britain Companion Specification |
| GFI | GIT For Industry |
| GIT | GBCS Interface Testing |
| GSME | Gas Smart Metering Equipment |
| HAN | Home Area Network |
| KRP | Known Remote Party |
| MAC | Message Authentication Code |
| NA | Not Applicable |

| Acronym | Description |
|---|---|
| OTA | Over-The-Air |
| RD | Reference Document |
| SCPI | Standard Commands for Programmable Instruments |
| SDB | Software Database |
| SMETS | Smart Metering Equipment Technical Specifications |
| SMKI | Smart Meter Key Infrastructure |
| SVF | System Validation Facility |
| TBC | To be confirmed |
| TBD | To be defined |
| UTRN | Unique Transaction Reference Number |
| ZCL | ZigBee Cluster Library |

**Table 20 – Acronyms.**