GSM Protocol Stack

GTPF - GPF Test Platform

User Guide

Author: Condat AG
Alt Moabit 91d
10559 Berlin
Germany

Date: 21-March-2002
ID: 8434.312.00.001

Table of Contents

0 Document Control

Condat AG
Alt Moabit 91d
10559 Berlin
Germany

Telephone: +49.30.39094-0
Fax: +49.30.39094-300
Internet: www.condat.de
E-mail: gsm@condat.de

0.1 Document History

ID	Author	Date	Status	Remarks
8434.312.00.001	JG	May 14, 02	Being Processed	Initial

0.2 References

[GSM 2.30]	ETS 300 511: July 1995 (GSM 02.30 version 4.13.0) Man-Machine Interface (MMI) of the Mobile Station (MS), ETSI
[FUG]	8434.100.01.001; Sep 07, 2001; Frame – User Guide (frame_users_guide.doc)
[PCO2]	8415.090.00.002; May 15, 2000; PCO2 – Tracing Environment (pco_userguide.doc)
[TCAL]	8434.312.00.001; Jan 31; 2001; TCAL – TAPCaller User Guide (tapcaller_userguide.doc)
[TCC]	8415.028.99.201; March 15, 2000; Test Case Control – User guide (8415_028.doc)
[XPAN]	8415.089.00.001, October, 2000, xPanel – MMI Test Application (PC) (xpan_userguide.doc)

0.3 Abbreviations

CMS	Condat Multitasking System
PCO	Point of Control and Observation
PS	Protocol Stack
TAP	Test Application Process
VCMS	Virtual CMS

0.4 Terms

GUI tools	MSCView, PCOController with GUI, PCOServer, PCOViewer, TAPCaller
Doc2text	Tool that converts MS Word documents into plain text files

1 Introduction

This documentation is meant for protocol stack testers that use the GPF test platform consisting of the new FRAME, TAP, PCO2, TAPCaller and MSCView. It is assumed that the user is familiar with basics of Condat's test tool chain. The authors tried to list all steps necessary for realising simple test sessions using the platform. In order to keep this document simple it is refrained from listing all possible options of the tools. For more detailed information on the usage of the tools the reader may have a look at the documentation referenced in the text below and the context sensitive help of the GUI tools.

Test cases and suites are used in order to test the implementation of protocol stacks. The Test Application Process (TAP) is the real test tool. The actions performed by TAP can be described in simple words as follows: It sends primitives to an entity of the PS, waits for an reaction of the PS (i.e. waits for a primitive from the entity under test) and compares the received primitive with the expected one.

TAPCaller is a graphical frontend for executing test cases. It should simplify the call of the TAP test tool.

PCO2 is a tool that finally provides an interface for tracking all kind of traces and primitives.

MSCVIEW is a tool used to display the expected structure of the flow of messages/primitives (from ~.tds files) or the sequence of messages/primitives of an already executed test case (from ~.dbg files).

The expression 'FRAME' is the summary of the generic protocol stack framework components in a protocol stack. This frame work consists of the components 'FRAME' for the real frame functionality and the test interface functionality including the test interface drivers.

2 Overview

The following illustration shows the GPF software architecture used for testing protocol stacks.

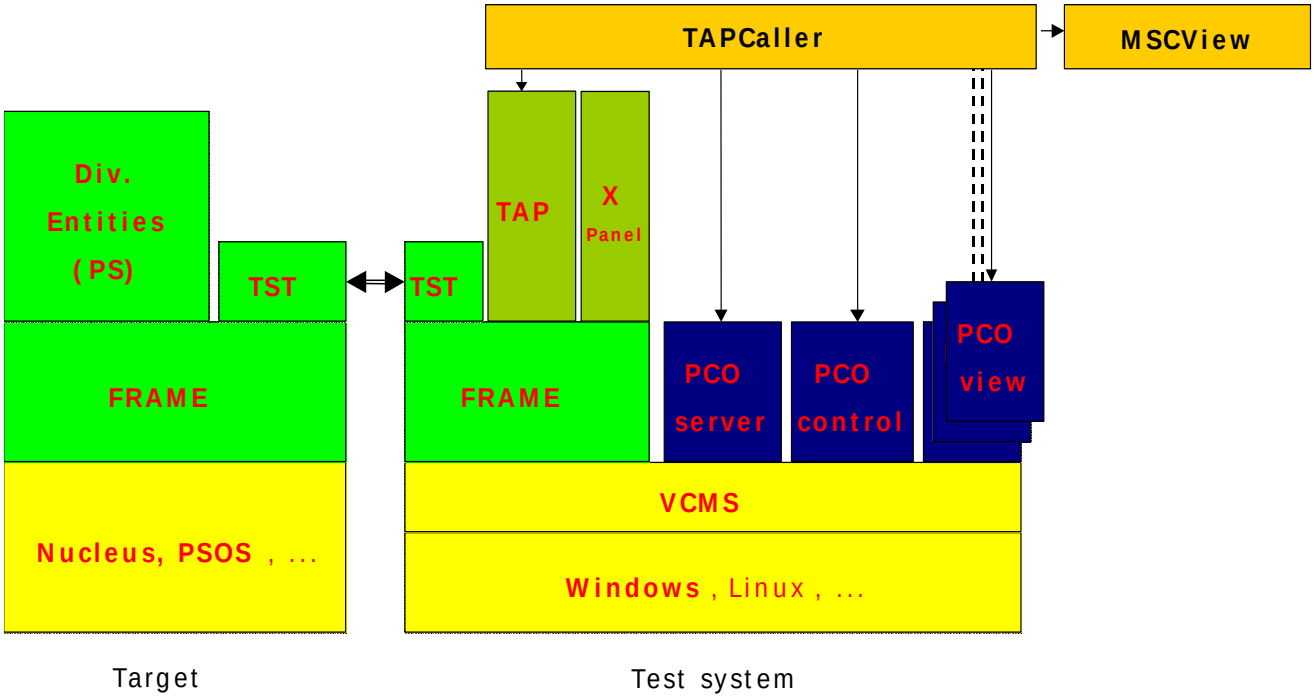

Figure 1: GPF software architecture

On the left side of the illustration the target, i.e. the system to be tested, is shown. It consists of an operating system (Nucleus, PSOS, ...), the FRAME, the test interface TST and several protocol stack entities. The FRAME, which abstracts from underlying operating systems and provides additional functionality, consists of several libraries, which are linked to the entities and the test tools. Therefore, the user who is only interested in testing does not have to care about the FRAME (see [FUG] for further information on the FRAME).

The test interface entity is running inside an independent Windows process. It abstracts from the real interface (simulated USART, USART or Socket interface), so that the user has a unique software interface for testing regardless of implementation details of the underlying hardware interface. TST is started automatically by the test tools whenever it is needed, but the user may change driver settings with an additional call of TST.

The entities of the target system are tested using the TAP, which stimulates these entities via the test interface TST. TAP sends primitives to an entity of the PS, waits for a reaction of the PS (i.e. waits for a primitive from the entity under test) and compares the received primitive with the expected one. Test cases are produced from so-called test documents. These documents are written in MS Word. Normally, each document contains the test cases for one PS entity. The user can build test cases using the scripts mktc, mkalltc and mkallin1tc.

xPanel provides a mobile-like keypad and a screen for display response. It can be used for stimulating PS entities, too (see [XPAN] for further information on xPanel).

The components of PCO2 (PCOServer, PCOController and PCOViewers) are used to track all traces and primitives exchanged via the test interface. The PCO server is the fundamental part of PCO2. It receives traces and redirected primitives from a testinterface-entity running on top of the FRAME. The main purpose of the PCOController is to change the status of a running PCO server and to send CONFIG-primitives to the stack via the server. There is a GUI (pco_ctrl.exe) and a command line based PCOController (pcoc.exe). The PCOViewer is WIN32-Viewer, available in two versions: with CCDEdit-support (pco_view.exe) and without (pco_miniview.exe). The first one needs the ccddata_dll.dll to run which has to be built with the current cdg-files (if not delivered as well). PCOViewers are used for formatted output of traces and primitives using several filters. Besides, it is also possible to export files in a format which can be read by MSCView (~.dbg files). As can be seen from the picture there is only one server and one controller concurrently running, but it is possible to execute several PCOViewers simultanously, e.g. with different filter settings.

MSCView can be used to display the expected flow of messages/primitives of a test case. For this purpose MSCView evaluates test definition scripts (~.tds files). For each test case a tds file is produced while generating executable test case DLLs. In addition, MSCView is able to show the time structure of already executed test cases from ~.dbg files exported by a PCOViewer.

TAPCaller is a graphical front-end to control all test tools. Its main purpose is to provide a user-friendly interface for the TAP. Besides, it can be used for logging flows of primitives using PCOServer, viewing test results with PCOViewers and it is possible to start MSCView.

VCMS is a version of the Condat Multitasking System (CMS) used for Windows NT 4.0. CMS is an operating system layer that supports the split of applications into several, independent and communicating processes and their quasi-parallel processing. VCMS is used to communicate between the frame and the PCO stuff. Since VCMS is implemented as a Dynamic Link Library the user does not have to care about it. The tools do all things necessary for the communication between them.

3 Necessary Libraries

For the subsequent build steps and test tool executions the following Dynamic Link Libraries are needed or are useful:

ccd.dll	functional part of CCD
cms.dll	CMS (Condat Multitasking System)
frame_wn.dll	the FRAME for Windows NT
ipc.dll	extension of CMS (needed by PCO)
misc_wn.dll	miscellaneous stuff (part of the FRAME)
moanbtn.dll	enables moan button for GPF GUI tools in the system tray (for complaints about these tools), not mandatory for well-functioning of the GUI tools
tif_wn.dll	test interface drivers

The DLLs can be found in GPF's binary directory (currently "\gpf\BIN"), except ipc.dll (in "\gpf\tools\bin"). For the debug versions of the frame libraries a "_db" is appended at the end of the file name, e.g. "frame_wn_db.dll". The debug versions of the libraries are in a subdirectory of the binary directory (currently "\gpf\BIN\debug", or "\gpf\tools\bin\debug" for "ipc.dll").

All other DLLs and executables that are needed for testing are mentioned in the text below.

4 Build steps

Requirements for all subsequent build steps: 4nt with a project specific Initvars.BAT already started

4.1 SAP/MSG documents and ccddata_dll.dll

SAP documents describe the structure of primitives interchanged between the entities of one PS. MSG catalogues specify the composition of messages that are sent from an entity to its peer via the air interface as specified by standardisation organisations. In order to use these messages and primitives within a PS or for the test tools, "makcdg.bat" has to be called. This batch file use doc2text, XGEN, CCDGen, make and a C compiler to produce include files and libraries.

CCDDATA is a library that represents the cdg tables and constants generated by CCDGEN. It must always be rebuilt if the interfaces defined in the SAP and MSG catalogues have changed. The CCDDATA library can be used by tools as well as the PS.

4.2 TAP2

TAP2 uses Ccddata_dll.dll as source that describes how to build up a message or primitive. Therefore, TAP2 does not need to be compiled and linked each time a MSG or SAP document is changed, only "Ccddata_dll.dll" has to be rebuilt. That is why a TAP2 executable "tap2.exe" can be found in GPF's binary directory.

4.3 Test Cases

Further documentation: [TCC]

When using TAP, test cases compiled as Dynamic Link Libraries are necessary. The description of the test cases are stored in MS Word documents. GPF provides three batch files for generating executable test cases. Two out of them are described here. In order to create one executable test case, the batch file MKTC.BAT is used. The following line has to be executed from a 4NT box:

MKTC –gen <Entity> <TestCase>

Example: MKTC –gen CC CC001

The batch file MKALLTC.BAT is provided to perform the preparing of all test cases of a test document with one command. The command is called from a 4NT box:

MKALLTC <Entity>

Example: MKALLTC RR

The batch files MKALLTC and MKTC use doc2text, TDSGen, MAKE and a C compiler in order to build test case DLLs.

5 Test Tools

5.1 TST

Required binaries for usage: TST executable

The test interface entity is running inside an independent Windows process. It is started by the test tools automatically, but the user may change the test interface driver or its settings by calling TST from command line. The following command line applies to TST:

TST interface [interface options]

TST is the name of the test interface executable, i.e. "tst.exe". Supported interfaces are simulated USART (interface="sim"), serial interface (USART, interface="com "+ comport number) and socket interface (interface="socket " + hostname).

Example: tst com 1

The option "-h" lists all other options for TST. This option can also be used for all command line tools described below.

5.2 TAP2

Required binaries for usage: TAP2 executable

PS executable

Ccddata_dll.dll

Compiled test cases

Further documentation: [TCC]

In its simplest form, the processes needed for a test are started from the command line of a 4nt Box. At first start a PS. In order to start TAP2 the following command line is to be used:

Tap2.exe [options] testcase[s]

Tap options can be used to change default settings, e.g. the interface (default: simulated USART). The parameter "testcases" consists of the directory where test case DLLs reside and a list of test cases that should be executed. For a short description of other command line options call TAP with the parameter "-h".

Example: tap2 z:\gsm\condat\ms\test\test_cc CC001

Unlike former versions, the TAP on new FRAME produces only one output file which contains the date and time of test execution as well as the test verdict. All other tracking and tracing is done by PCO2 components.

5.3 PCO

Required binaries for usage: PCO executables (pco_srv.exe, pco_view.exe, pco_ctrl.exe/pcoc.exe)

PS executable

	xPanel or TAP2 executable for stimulating the PS
	Ccddata_dll.dll
	Compiled test cases when used with TAP2
Further documentation:	[PCO2]

PCO is an independent tool which can be used without TAP, e.g. with xPanel. As mentioned before, PCO consists of three parts: PCOServer, PCOController and PCOViewer. The user needs to start a PCOServer, if he or she wants to do anything with PCO. There are three main application scenarios, which are described in this document.

5.3.1 Viewing flow of traces and primitives

Start the PCOServer and a PCOViewer. Start the protocol stack. Start TAP with the test case that should be executed or xPanel to stimulate the protocol stack. You will see the flow of test data in your PCOViewer.

5.3.2 Logging test data

- Using the command line PCOController: Start a PCOServer and a 4nt box. Call "pcoc.exe" (the command line PCOController) from your 4nt box with the parameter "-start <testname>" to start a test session named "testname". Then you can start TAP or stimulate the running protocol stack with xPanel. Call "pcoc –stop" at the end of your test session.

  You may change the test session path with the command line option "-spath <path>" before starting the test session. The log files are written to the specified directory.

- Using the controller with a GUI: Start a PCOServer and "pco_ctrl.exe" (the PCOController with GUI). Switch to "Online mode" with the uppermost button if necessary. Type a name for your test session in the provided edit box. Use the "Start logging" button to trigger the logging of test data. Stimulate the protocol stack with TAP or xPanel. The button "Stop logging" is used to terminate the test session.

All test data logged during the test session can be found in a file named after the test session name with the file extension ".pco". To view these results later follow the instructions in the next section.

Viewing flow of traces/primitives and logging test data can be done together.

5.3.3 Viewing already logged test data

- Using the command line controller: Start a PCOServer, a PCOViewer and a 4nt box. Call "pcoc.exe" in your 4nt box with the parameter "-open <testname>" to load a test session. After executing "pcoc –distrib" the primitives and traces from the log file are presented in the PCOViewer. Call "pcoc.exe" with the parameter "-close" to quit viewing test data in the PCOViewer.

- Using the controller with a GUI: Start a PCOServer, a PCOViewer and the PCOController ("pco_ctrl.exe"). Switch to "Replay mode" with the uppermost button if necessary. Select a session name and push “Replay”. Connected PCOViewers will show the logged traces/primitives.

It is possible to do all the things described in this chapter with the TAPCaller as well (see chapter "5.4.3 Using PCO2 application scenarios within TAPCaller").

Besides, with the GUI-PCOController the user is able to create test environments, i.e. a list of applications specified in an ASCII-file will be started when starting the controller. This may simplify the call of the different tools when used more than once (see [PCO2] for further information).

5.4 TAPCaller

Required binaries for usage: TAPCaller executable (tapcaller.exe)

TAP2 executable and compiled test cases

PS executable

Ccddata_dll.dll

PCO executables if desired

Further documentation: [TCAL]

5.4.1 TAPCaller behaviour while starting

Since the PCO2 is not part of the TAP, these tools have to be handled independently by TAPCaller. While starting TAPCaller it is therefore tested whether a PCOServer is already running. If not, it is started from GPF's binary directory of the current working drive. The user is able to avoid starting PCOServer with the command line parameter "-nopco".

5.4.2 Necessary steps to start a test case

Start TAPCaller. Load test case DLLs using the dialog accessible with the menu item <Configuration><Selection>. Using the menu option "<Configuration><Settings>" the dialog box "Settings" is displayed.

There the user has to specify the location of the appropriate TAP executable and the interface which stimulates the protocol stack (default: simulated USART) on the sheet "TAP Options" as well as the location of the PS executable on the sheet "Test Tools". If the user wants start the PS himself/herself the checkbox "Enabled" has to be left open.

Leave the dialog and start test cases using the menu items <Control><Start test> (starts all tests in the main window) or <Control><Start selected tests> (starts only selected test cases).

5.4.3 Using PCO2 application scenarios within TAPCaller

In order to work correctly TAP and PCO have to be started from the same directory. Therefore, whenever the path of TAP is changed the user is asked whether TAPCaller should change the directory for the PCO executables, too. It is recommended to accept this automatic configuration. Then the old PCOServer is stopped and a new one from the new TAP directory is started.

5.4.3.1 Viewing flow of traces and primitives

Configure TAPCaller as described in subsection "5.4.2 Necessary steps to start a test case".

In order to watch all traces/primitives while executing a test case, the user has to activate PCOViewer configurations. This is possible with the sheet "Test Tools" of the menu item <Configuration><Settings>. In the centre of this dialog the user can choose among a list of PCOViewer configurations which can be

found in the directory above this list. At least he or she is able to select the "Standard Viewer". With this selection the PCOViewer is called without any parameters. Then, the last configuration used on this machine is loaded by PCOViewer.

Leave the dialog and start test cases. For each test case PCOViewers with the selected configurations are started. It terminates at the end of the test case. If the user wants to have a look at the traces/primitives longer he or she may use the "H"-button of TAPCaller's toolbar. This stops the sequence of test case execution at the end of each case.

5.4.3.2 Logging test data

Configure TAPCaller as described in subsection "5.4.2 Necessary steps to start a test case".

In order to log all test data the sheet "Test Tools" of the menu item <Configuration><Settings> has to be used. The check box "Log with PCO while testing" has to be activated. For each test case a log file is created that is stored in the same directory as the file with the test verdict. The logging is started and stopped automatically at the beginning and termination of each subsequent executed test case.

Viewing flow of traces/primitives and logging test data can be done together.

5.4.3.3 Viewing already logged test data

Beside test verdict files it is possible to access already created PCO log files with TAPCaller. The user does not need a TAP and a PS executable for this purpose, but a list of test cases, a PCOServer and a list of PCOViewer configurations.

Load test cases with the dialog of the menu item <Configuration><Selection>. A PCOServer is mostly already started. In order to choose among a list of PCOViewer configurations, the user has to open the sheet "Test Tools" of the menu item <Configuration><Settings>. In the centre of this dialog the list with at least one entry "Standard Viewer" appears. With this selection the PCOViewer is called without any parameters. Then, the last configuration used on this machine is loaded by PCOViewer.

Leave the dialog and click with the right mouse button on the test case for which the PCO log file should be presented. A context menu appears. It contains an entry for the verdict file and one entry for each selected PCOViewer configuration if the respective files are found (the verdict and the PCO log file). With a click on an appropriate entry a PCOViewer with the chosen configuration is started and the PCO log file is loaded.

5.5 Crash, malfunction, unexpected behaviour

Developers of the described tools tried to foresee possible error situations. Therefore each tool report errors and their causes, if known. Generators produce error files, GUI programs use message boxes and so on.

Nonetheless, there are situations where no tool is able to determine the reason for failure. Listed below are common errors and possible solutions.

- Since many tools use DLLs, the dependencies between them should be resolved. Besides, a missing DLL could be a problem. Have a look at the list in chapter "3 Necessary Libraries" or view the dependencies with an appropriate tool. This may give hints to a solution.
- Have a look at the documents referenced above. Maybe the strange behaviour of the tools is intentional :-)

- Close all tools and restart them.
- After crashes there are sometimes seemingly active processes in the task list. Use the task manager in order to look for processes that are not working anymore.
- Sometimes Windows needs a reboot ;-)
- If nothing else helps, ask the developers of the tools.