Technical Document

GSM PROTOCOL STACK

GPF

TCC – TEST CASE CONTROL

USER GUIDE

Document Number:	06-03-30-UDO-0001
Version:	0.10
Status:	Draft
Approval Authority:	
Creation Date:	1999-Mar-29
Last changed:	2015-Mar-08 by
File Name:	8415_028.doc

Important Notice

Texas Instruments Incorporated and/or its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products, software and services at any time and to discontinue any product, software or service without notice. Customers should obtain the latest relevant information during product design and before placing orders and should verify that such information is current and complete.

All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment. TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI products, software and/or services. To minimize the risks associated with customer products and applications, customers should provide adequate design, testing and operating safeguards.

Any access to and/or use of TI software described in this document is subject to Customers entering into formal license agreements and payment of associated license fees. TI software may solely be used and/or copied subject to and strictly in accordance with all the terms of such license agreements.

Customer acknowledges and agrees that TI products and/or software may be based on or implement industry recognized standards and that certain third parties may claim intellectual property rights therein. The supply of products and/or the licensing of software does not convey a license from TI to any third party intellectual property rights and TI expressly disclaims liability for infringement of third party intellectual property rights.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products, software or services are used.

Information published by TI regarding third–party products, software or services does not constitute a license from TI to use such products, software or services or a warranty, endorsement thereof or statement regarding their availability. Use of such information, products, software or services may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Change History

Date	Changed by	Approved by	Version	Status	Notes
1999-Mar-29	VK et al.		0.1		1
1999-Mar-31	MS et al.		0.2		2
1999-Aug-31	VK et al.		0.3		3
1999-Nov-01	HJS et al.		0.4		
1999-Dec-15	MS et al.		0.5		4
2000-Jan-21	HJS et al.		0.6		5
2000-Mar-15	HSC et al.		0.7		6
2000-Jul-10	HJS et al.		0.8		7

2002-Jan-15	HSC et al.		0.9		8
2003-May-20	XINTEGRA		0.10	Draft	
2003-Sep-08	HSC		0.11		9

Notes:

1. Initial version
2. English check/Format check
3. Add to section “Prerequisites
4. New features added/existing features amended
5. New Template (MS)
6. Updated
7. Format/English Check (MS)
8. Updated
9. New document number

Table of Contents

List of Figures and Tables

List of References

[ISO 9000:2000] International Organization for Standardization. Quality management systems - Fundamentals and vocabulary. December 2000

1.1 References

[C_6147.302] 6147.302.99.111; March 30, 1999
Test Case Script Language User Guide; Condat

1.2 Abbreviations

G23	In the test environment: the implementation of G23 on a PC
EUT	Entity Under Test
ISS	Infrastructure Simulation
IUT	Implementation Under Test
MFW	MMI Framework
MMI	Man Machine Interface
MSC	Message Sequence Chart
NT	Windows NT
PCO	Point of Control and Observation
PS	Protocol Stack
TAP	Test Application Process
TCSL	Test Case Script Language
TDL	Test Definition Language

2 Introduction

G23 is a software package implementing Layers 2 and 3 of the ETSI-defined GSM air interface signaling protocol, and as such represents the part of a GSM mobile station's protocol software which is both, platform and manufacturer independent. Therefore, G23 can be viewed as a building block providing standardized functionality through generic interfaces for easy integration.

The G23 suite of products consists of the following items:

- Layers 2 and 3 for speech & short message services,
- Layers 2 and 3 for fax & data services,
- Application Control Interface,
- Slim MMI [02.30] and
- Test and integration support tools.

This document describes how to define, create, run and evaluate test cases for the Condat GSM Protocol Stack G23. Test cases come in two forms:

- Module Test Cases
- Multi Layer Test Cases

Both of these types of test cases are covered in this document. The following table presents an overview of these two kinds of tests:

	Module Test	**Multi Layer Test**
Base	Specification and Implementation (SAP, MSC, SDL, C-Code)	GSM 11.10
Attribute	single entity, white box test, primitive - oriented variants allowed	multiple entities, black box test, message - oriented, variants allowed
Definition	TDL, WinWord document	TCSL, WinWord document
Creation	TdsGen/C-Compiler	TdsGen/C-Compiler
Executable	Test DLL	Test DLL
Environment	PC implementation of G23, Nucleus MNT, TAP	PC implementation of G23, Nucleus MNT, TAP, ISS
Execution Tools	TapCaller	TapCaller

Both kinds of tests are defined in WinWord documents. Executable Test DLLs are generated from these documents by applying the TdsGen generator and a C-Compiler. A Test DLL is created for each test case.

Test cases are one type of deliverable created in the Condat software process. The following list contains further deliverables described elsewhere:

- Message Sequence Chart Definitions
- SDL Specification
- Service Access Point (SAP) Specification
- Air Interface Messages Definition
- Implementation

3 Prerequisites

Certain assumptions are made related to the test environment:

- The IUT (implementation under test) has been installed
- Microsoft Developer Studio 97 or higher has been installed
- Microsoft WinWord 97 or higher has been installed
- 4NT Version 2.50 or 3.01 or higher has been installed
- Environment variables have been set (call INITVARS)

The following tools and files are used (directly or indirectly) within the test case process:

- TUCT_TECH.DOT (create test case)
  Á WinWord Document Template file used for test documents containing several useful macros..
- initvars.bat (create and execute test case)
  A simple batch file to set up environment variables.
- tap2_%PROST%.exe (execute test case)
  Command line application. The Test Application Process takes the test case as a parameter and executes it. The environment variable %PROST% (**pro**tocol **st**ack) is set by initvars.bat (see section 4.1)
- tapcaller.exe (execute test case)
  MS Windows application used for regression testing. TapCaller will activate TAP and the IUT (implementation under test).
- testdll.mk (create test case)
  Makefile needed to create test case DLLs from the TDS files which contain the test case in an intermediate description.
- DOC2TXT.EXE (create test case)
  This tool is applied on the WinWord file containing the test case description. A conversion to a raw text representation is carried out.
- tdsGen.exe (create test case)
  Test Case Generator. This generator is applied on the raw text representation of the test cases. An intermediate description, TDS files, are created.
- gnumake.exe (create test case)
  This Make-Program is called by MKTC.bat resp. mkalltc.bat and is controlled by TestDll.mk.
- MKTC.bat (create test case)
  Batchfile that generates a dedicated test case DLL.
- mkalltc.bat (create test case)
  Batchfile that generates all test case DLLs belonging to a test case document.

4 Test Case Definition

As stated previously, two types of test cases exist: Module Tests and Multi Layer Tests. The difference between these is expressed in some aspects of the test case definition. Module Tests are defined in terms of the Test Definition Language (TDL), whereas Multi Layer Tests are defined in terms of the Test Case Script Language (TCSL).

There are some common properties of both Module Tests and Multi Layer Tests. The creation and top level structuring of test case documents are equal for both types of test cases.

4.1 Creating and editing test case documents

Test cases are defined in documents created and edited by Microsoft WinWord 97. It is recommended to chose the following options in the Microsoft WinWord 97 program:

Tools|Options|Edit	Use smart cut and paste	Not marked
	Tabs and backspace set left indent	Not marked
Tools|AutoCorrect|AutoCorrect	Capitalize first letter of sentences	Not marked
Tools|AutoCorrect|AutoFormat as you type	"straight quotes" with "smart quotes"	Not marked
	Bold and _underline_ with real formatting	Not marked
Tools|AutoCorrect|AutoCorrect	In the list delete the entries for '<==' and '==>'	

For the test case generator to work properly, it is necessary to place test case documents in the directory %TSTDOCDIR%.

It is recommended to place all test cases related to one entity of the Condat Protocol Stack G23 into one test case document.

Test case documents must use the template file TUCT_TECH.DOT. This DOT file contains the WinWord macros used to process test case documents.

A new test case document may be created in one of two ways:

- by copying and modifying an existing document
- by using the DOT template file TUCT_TECH to add test cases

It is recommended to use the first method (copy/modify).

Microsoft WinWord 97 is used to edit test case documents. In order to execute the test case generator, the environment variables defined in the batch file INITVARS (e.g. TESTROOT, SIDE) must be defined when WinWord 97 is used to edit test case documents. This can be ensured in one of the following ways:

- by calling initvars from within an 4NT box, %WINWORD% can be called subsequently (see 5.1.2)
- by placing CALL initvars in autoexec.bat
- by adding the environment variables to the NT environment

It is recommended to use the first method to set up the environment variables. The batch file initvars.bat needs four parameters:

1. The name of the protocol stack to be tested (e.g. "gprs"). The environment variable %PROST% is set with this name.
2. The string "ms" or "bs" depending on if it is a **m**obile **s**tation or a **b**ase **s**tation stack to be tested.
3. The drive of the local test development (e.g. "C:").
4. The base directory of the local test development (e.g. "\GSM\Condat").

For example:

initvars gprs ms z: \GSM\Condat

4.2 Structure of test case documents

The structure of test case documents is given by the top level headings of the test case document:

Cover page
0 Table of Contents
1 Document Control
2 Parameters
3 Test Cases
3.1 Heading of the first test case group
3.1.1 <DocName>000: Heading of the first test case
History
3.1.2 <DocName>001: Heading of the second test case
History
…
3.2 Heading of the second test case group
3.2.1 <DocName>nnn: Heading of the nnnth test case
History
…

where <DocName> is the name of the test case document (e.g. MCC).

This structure must be followed in order to allow the test case generator to perform an evaluation of the test case document. The test case generator is sensitive to text which is underlined in the previous example. The first generator sensitive text of a test case document is the heading "2 Parameters" (note the plural form). This section contains the test data of the test case document. The structure of this test data is described in the subsequent sections of this document.

The next section of a test case document contains test cases. This section is introduced by the heading "3 Test Cases" (also plural). The test cases are grouped. A test group is introduced by a heading at level 2 (e.g. "3.1 Heading of the first test case group "). However, this heading is not evaluated by the test case generator.

The beginning of a test case definition is identified by a heading at level 3 (e.g. "3.1.1 MCC000: Initialization"). The end of a test case definition is identified by the string "History". The structure of the test case contents is described in the subsequent sections of this document.

4.3 Module Test Cases

4.3.1 Purpose

Module Test Cases are used to test GSM entities (e.g. RR). The Test Definition Language (TDL) is used to define Module Test Cases. The structure of test case documents containing Module Test Cases is dominated by the two top level sections "Parameters" and "Test Cases".

4.3.2 Parameters Section

The Parameters Section defines test data which is used in the section "Test Cases". TDL provides the following constructs to define named test data:

- BYTE define an 8 bit data item
- SHORT define a 16 bit data item
- LONG define a 32 bit data item
- DECLARATION define a handle
- FIELD/ENDFIELD define an array of bytes (**OBSOLETE**!)
- BEGINARRAY/ENDARRAY define an array of bytes (instead of FIELD)
- BEGINARRAY_PART/ENDARRAY define a partial array of bytes
- BEGIN_SHORT_ARRAY/ENDARRAY define an array of 16 bit data items
- BEGIN_SHORT_ARRAY_PART/ENDARRAY define a partial array of 16 bit data items
- BEGIN_LONG_ARRAY/ENDARRAY define an array of 32 bit data items
- BEGIN_LONG_ARRAY_PART/ENDARRAY define a partial array of 32 bit data items
- SET_BITBUF/ENDBITBUF define a buffer for a bit field longer than 24 bits
- SET_SDU/ENDSDU define a buffer for an sdu
- BEGIN_MSTRUCT/ENDSTRUCT define a structured Information Element
- BEGIN_PSTRUCT/ENDSTRUCT define a structure in a Primitive
- SET_COMP initialize a component of a structure
- SKIP_COMP ignore a component of a structure
- SHOW_COMP like SKIP_COMP but display the content on arrival
- FORBID_COMP signal an error if the structure component is received
- BEGIN_PSTRUCT_ARRAY/ENDARRAY define an array of structures (**OBSOLETE**!)
- BEGIN_STRUCT_ARRAY/ENDARRAY define an array of structures (instead of BEGIN_PSTRUCT_ARRAY)
- BEGIN_STRUCT_ARRAY_PART/ENDARRAY define a partial array of structures
- STRING define a string
- NEED includes the following file
- NUM_ELEMENTS delivers the number of elements in data declared before

These language elements are referred to as macros and are described in the subsequent sections.

4.3.2.1 BYTE, SHORT, LONG

Description: The macro BYTE/SHORT/LONG is used to define a named 8/16/32 bit test data item.

Syntax: BYTE <TDS_NAME> <VALUE>

<TDS_NAME>: The name of the test data item. The name may be used in the section "Test Cases" or within the section "Parameters" to define more complex structured test data. The name must be unique within the <TDS_HANDLE> of all DECLARATION and STRING macros and the <TDS_NAME> of all BYTE, SHORT, and LONG macros respectively.

<VALUE>: An 8/16/32 bit data item, either in decimal or hexadecimal notation. A hexadecimal is prefixed by "0x".

Example:
```
BYTE    ARFCN                 0x52
SHORT   ACC_CTRL_CLASS_0000   0x0000
LONG    TMSI_BIN              0x00234534
```

4.3.2.2 DECLARATION

Description: The macro DECLARATION is used to define a handle which may be used in subsequent definitions of test data (e.g. BEGIN_PSTRUCT).

Syntax: DECLARATION (<TDS_HANDLE>)

<TDS_HANDLE>: The name of the handle. The name must be unique within the <TDS_HANDLE> of all DECLARATION and STRING macros and the <TDS_NAME> of all BYTE, SHORT, and LONG macros respectively. All DECLARATION macros must appear before the first macro SET_BITBUF or a macro with a name beginning with 'BEGIN' respectively.

Example: DECLARATION (MS_CLASS)

4.3.2.3 FIELD/ENDFIELD

Description: The macro FIELD is used to define an array of bytes (8 bit data items).
Warning: Do not use FIELD/ENDFIELD. It won't be supported in future versions. Use BEGINARRAY/ENDARRAY instead.

4.3.2.4 [BEGINARRAY[_PART]|BEGIN_SHORT_ARRAY[_PART]|BEGIN_LONG_ARRAY[_PART]]/ENDARRAY

Description: The macro BEGINARRAY is used to define an array of bytes (8 bit data item).
The macro BEGIN_SHORT_ARRAY is used to define an array of shorts (16 bit data item).
The macro BEGIN_LONG_ARRAY is used to define an array of longs (32 bit data item).
The variants of those three arrays with the suffix _PART are used to define partial arrays. During test case execution the TAP checks the supplied data against its definition in the corresponding SAP or MSG catalogue. For arrays of fixed length this means that the length is checked and the test will fail if it is not correct. With partial arrays it is possible to explicitly avoid this check and to define fewer array element than required.

Syntax: BEGINARRAY (<TDS_HANDLE>, <NUM_ITEMS>) ... ENDARRAY
BEGIN_SHORT_ARRAY (<TDS_HANDLE>, <NUM_ITEMS>) ... ENDARRAY
BEGIN_LONG_ARRAY (<TDS_HANDLE>, <NUM_ITEMS>) ... ENDARRAY

<TDS_HANDLE>: the name of the array. The name must be unique within the <TDS_HANDLE> of all DECLARATION and STRING macros and the <TDS_NAME> of all BYTE, SHORT, and LONG macros respectively.
<NUM_ITEMS>: The number of bytes/shorts/longs between BEGIN... and ENDARRAY.

The macro ENDARRAY does not take any parameters. Between BEGIN... and ENDARRAY, the bytes/shorts/longs must be specified, separated by commas, in decimal or in hexadecimal notation. A hexadecimal is prefixed by "0x".

Example:
```
BEGINARRAY (IDENT_DIGITS_IMSI, 12)
    2, 6, 2, 1, 3, 4, 3, 2, 1, 0, 9, 7
ENDARRAY
BEGIN_SHORT_ARRAY_PART (COUNTS, 3)
    900, 300, 1200
ENDARRAY
BEGIN_LONG_ARRAY (TIMERS, 3)
    90000L, 3000L, 12000L
ENDARRAY
```

4.3.2.5 SET_BITBUF/ENDBITBUF

Description: The macros SET_BITBUF/ENDBITBUF are used to define a named buffer for a bit field longer than 24 bits. Such bit fields occur in Air Interface Messages only.

Syntax: SET_BITBUF (<NAME>, <TDS_HANDLE>, <NUM_BITS>) ... ENDBITBUF

<NAME>: the name if the bit field within a Information Element. The name must be specified in quotation marks.
<TDS_HANDLE>: the name of the test data item. The name must be unique within the <TDS_HANDLE> of all DECLARATION and STRING macros and the <TDS_NAME> of all BYTE, SHORT, and LONG macros respectively. The name

may be used within the section "Parameters" to define more complex structured test data.

<NUM_BITS>: the length of the bit field. This number must not be greater than the maximal size of the bit field in decimal or in hexadecimal notation.
A hexadecimal is prefixed by "0x".

The macro ENDBITBUF does not take any parameters. Between the macros SET_BITBUF and ENDBITBUF, the data for the bit field is specified as bytes in decimal or in hexadecimal notation. A hexadecimal is prefixed by "0x". The bytes are separated by commas. The number of bytes needed is (<NUM_BITS>+7)/8.

Example:
```
SET_BITBUF ("tmsi", TMSI_BIN_1, 32)
    0,0x23, 0x45, 0x34
ENDBITBUF
```

4.3.2.6 SET_SDU/ENDSDU

Description: The macros SET_SDU/ENDSDU are used to define a buffer for an sdu.

Syntax: SET_SDU (<TDS_HANDLE>, <NUM_BITS>,<OFFSET>) … ENDSDU

<TDS_HANDLE>: the name of the test data item. The name must be unique within the <TDS_HANDLE> of all DECLARATION and STRING macros and the <TDS_NAME> of all BYTE, SHORT, and LONG macros respectively. The name may be used within the section "Parameters" to define more complex structured test data.

<NUM_BITS>: the length of the bit field. This number must not be greater than the maximal size of the bit field in decimal or in hexadecimal notation.
A hexadecimal is prefixed by "0x".

<OFFSET>: an offset (in bits) with that the sdu shall be written into the buffer.

The macro ENDSDU does not take any parameters. Between the macros SET_SDU and ENDSDU, the data for the bit field is specified as bytes in decimal or in hexadecimal notation. A hexadecimal is prefixed by "0x". The bytes are separated by commas.

Example:
```
SET_SDU (SDU_U_START_CC, 32, 0)
    0,0x23, 0x45, 0x34
ENDSDU
```

4.3.2.7 BEGIN_MSTRUCT/ENDSTRUCT

Description: The BEGIN_MSTRUCT macro identifies the start of an Information Element (IE) definition. An IE is part of an Air Interface Message definition. In TDS, Air Interface Messages or ABIS messages are denoted by using the "sdu" language element (ref. section "Test Cases"). If a message contains a high layer message (e. g an ABIS message contains an RR message), the high layer messages is denoted by using the "hl_sdu" language element.

Syntax: BEGIN_MSTRUCT ("<INFO ELEMENT>", <TDS_HANDLE>) … ENDSTRUCT

<INFO ELEMENT>: the name of the information element as given in the Air Interface Message definition document.

<TDS_HANDLE>: a name which denotes the test data for the information element.
This name is used in the section "Test Cases" to refer to the test data. The handle <TDS_NAME > must have been defined with the DECLARATION macro.

The macro ENDSTRUCT does not take any parameters.
Between the macros BEGIN_MSTRUCT and ENDSTRUCT, the test data for the Information Elements are defined by using the macros SET_COMP and SKIP_COMP. All parameters of the information element INFO_ELEMENT mentioned in the message catalogue must appear in the same order as (first) parameter of an SET_COMP or SKIP_COMP macro respectively.

Example:
```
BEGIN_MSTRUCT ("chan_desc", CHANNEL_DESC_XYZ)
    SET_COMP ("chan_type", SDCCH_4_S1)
    SET_COMP ("tn", TN_4)
    SET_COMP ("tsc", TSC_2)
    SET_COMP ("hop", HOP_NO)
    SET_COMP ("arfcn", ARFCN_800)
    SKIP_COMP ("maio")
    SKIP_COMP ("hsn")
ENDSTRUCT
```

4.3.2.8 BEGIN_PSTRUCT/ENDSTRUCT

Description: The BEGIN_PSTRUCT macro identifies the start of a Primitive Component definition. A Primitive Component is part of a Primitive.

Syntax: BEGIN_PSTRUCT ("<PRIM COMPONENT>", <TDS_HANDLE >) ... ENDSTRUCT

<PRIM COMPONENT>: the name of the Primitive Component as given in the Service Access Point definition document.

<TDS_HANDLE>: a handle which denotes the test data for the Primitive Component. This name is used in the section "Test Cases" to refer to the test data or it is used within definition of other Primitive Components. The handle <TDS_HANDLE> must have been defined with the DECLARATION macro.

The macro ENDSTRUCT does not take any parameters.
Between the macros BEGIN_PSTRUCT and ENDSTRUCT, the test data for the Primitive Components are defined by using the macros SET_COMP and SKIP_COMP. All parameters of information element INFO_ELEMENT mentioned in the Service Access Point description document must appear in the same order as (first) parameter of an SET_COMP or SKIP_COMP macro respectively.

Example:
```
BEGIN_PSTRUCT ("class", CLASS_MS)
    SET_COMP ("pclass", CLASS_2)
    SKIP_COMP ("pclass2")
ENDSTRUCT
```

4.3.2.9 SET_COMP

Description: SET_COMP is used to initialize an Information Element or a Primitive Component.

Syntax: SET_COMP (< NAME>, <VALUE>)

<NAME>: the name of a bit field within an Information Element or a Component of a Primitive. The name must be specified in quotation marks.

<VALUE>: the value of the test data item If <NAME> denotes an atomic element, then <VALUE> may be the <TDS_NAME> of a BYTE, SHORT or LONG macro. If for this name a value table is defined in the Service Access Point document or Air Message document, then the symbols of these c-macros may used also as <VALUE>.
If <NAME> denotes an array, then <VALUE> must be <TDS_HANDLE> of a BEGINARRAY, BEGIN_SHORT_ARRAY, BEGIN_LONG_ARRAY, BEGIN_MSTRUCT or BEGIN_PSTRUCT macro respectively.
IF <NAME> denotes a bit field with a length greater than 24 bits, then <VALUE> should be the <TDS_HANDLE> of a SET_BITBUF macro.

Example: SET_COMP ("tsc",TSC_2)

4.3.2.10 SKIP_COMP, SHOW_COMP

Description: SKIP_COMP and SHOW_COMP are used to indicate that an Information Element or a Primitive Component should not be included in the test data. Nevertheless in the case of SHOW_COMP the corresponding item received by the protocol stack will be displayed. In sent structures SHOW_COMP will act like SKIP_COMP.

Syntax: SKIP_COMP (< NAME>)

<NAME>: the name of a bit field within an Information Element or a Component of a Primitive. The name must be specified in quotation marks.

Example: SKIP_COMP ("hsn")
SHOW_COMP (“tp_oa”)

4.3.2.11 FORBID_COMP

Description: FORBID_COMP is used to indicate that an Information Element or a Primitive Component must not be included in the received test data. In sent structures FORBID_COMP will act like SKIP_COMP.

Syntax: FORBID_COMP (< NAME>)

<NAME>: the name of a bit field within an Information Element or a Component of a Primitive. The name must be specified in quotation marks.

Example: FORBID_COMP ("hsn")

4.3.2.12 BEGIN_PSTRUCT_ARRAY/ENDARRAY

Description: The BEGIN_PSTRUCT_ARRAY defines a primitive component which is described as an array of structures in
the Service Access Point document.

Warning: Do not use BEGIN_PSTRUCT_ARRAY. It won't be supported in future versions. Use BEGIN_STRUCT_ARRAY instead.

4.3.2.13 BEGIN_STRUCT_ARRAY[_PART]/ENDARRAY

Description: The BEGIN_STRUCT_ARRAY defines an array of structures.
The variant with the suffix _PART defines a partial array (see section 4.3.2.4) of structures.

Syntax: BEGIN_STRUCT_ARRAY(<TDS_HANDLE>,<NUM_STRUCTS>)
ENDSTRUCT

<TDS_HANDLE>: the name of the array of structures to be defined. The handle <TDS_HANDLE> must have been defined with the DECLARATION macro.
BEGIN_PSTRUCT_ARRAY and ENDSTRUCT enclose the elements of the array. Each element must be of the type PSTRUCT (BEGIN_PSTRUCT .. ENDSTRUCT).

<NUM_STRUCTS>: the count of structures belonging to the array. In the case of an array of variable length, only the valid entries are needed.

The macro ENDARRAY does not take any parameters.

Example:
```
BEGIN_PSTRUCT_ARRAY (CH_NUM_ARR_1,3)
   CH_NUM_1,
   CH_NUM_2,
   CH_NUM_3
ENDARRAY
```

4.3.2.14 STRING

Description: The STRING macro can be used to define a string. A string is internally stored as an array of bytes, but in contrast to BEGINARRAY it is defined as human readable character string and not byte by byte. A second difference is that the TDS_HANDLE of the string is declared implicitly by the STRING construct and not by a DECLARATION. Therefore each STRING must appear before any SET_BITBUF and before any macro beginning with 'BEGIN'.

Syntax: STRING (<TDS_HANDLE>,<CHAR_STRING>)

<TDS_HANDLE>: the name of the string to be defined. The name must be unique within the <TDS_HANDLE> of all DECLARATION and STRING macros and the <TDS_NAME> of all BYTE, SHORT, and LONG macros respectively.

<CHAR_STRING>: the string itself

Example: STRING (C_PLUS_CPAS, "AT+CPAS ")

4.3.2.15 NEED

Description: The NEED statement provides a simple mechanism to include (header) files, which may contain constants used in the test cases.

Syntax: NEED <FILE>

<FILE>: the name of the file to be included. Note that the file name is not wrapped in quotation marks.

Example: NEED foo.def

4.3.2.16 NUM_ELEMENTS

Description: The NUM_ELEMENTS delivers the number of elements in data declared before.

Syntax: NUM_ELEMENTS (<TDS_HANDLE>)

<TDS_HANDLE>: identifies the data of what the number of elements is delivered.

Example:
```
STRING(STRING_1, "Hello")
BEGIN_PSTRUCT (..., STRING_STRUCT)
   SET_COMP ("s_len", NUM_ELEMENTS(STRING_1))
   SET_COMP ("string", STRING_1)
ENDSTRUCT
```

4.3.3 Test Cases Section

The section headed "Test Cases" contains groups of test case definitions. A test case group contains any number of test cases. Each test case is defined in a separate section with the following structure:

- Heading (Level 3)
- Description
- Preamble
- Variants
- Primitive Sequence Chart
- Parametrization
- History

An exception is the first test case that is used for initialization purposes. It contains redirection commands. These redirection commands are placed into the block of the Message Sequence Chart (ref following).

4.3.3.1 Heading

The Heading of a Module Test Case must have the following structure:

n1.n2.n3 <test case name >: <one line text of the heading>

In this structure n1, n2 and n3 are decimal numbers. The test case name consists of the name of the test document and a three or four digits (e.g. l2r001). If the test document is divided into several parts (e.g. the test cases for GRR are in the documents GRR1.doc and GRR2.doc) the number at the end

of the file name can be left out (e.g. GRR1 can contain the test case GRR001). The test case name has to be unique among the documents belonging together. It is suggested to leave gaps in the numbering of test cases to make it easier to fill in further test cases later without renumbering the existing cases. Here is a valid example of a test case heading:

3.2.10 RR027: BCCH carrier not suitable (cell barred, with SIM)

4.3.3.2 Description

Each Module Test case contains a description of the actions performed. This description is specified in prose after the keyword "Description:". For example:

Description: The SYS INFO messages have reported that the BCCH carrier is not suitable as the cell is barred. ...

4.3.3.3 Preamble

Each Test Case may have a predecessor - a test case which is executed before the execution of the test containing the predecessor starts. The predecessor is specified by the keyword "Preamble:". If the test case has no predecessor then the line of text "Preamble: None" must be issued. Here is an example for using the keyword Preamble:

Preamble: RR012A

Note that a test case has at most one Preamble.

4.3.3.4 Variants

Similar test cases differing slightly in their parametrization and/or preamble may be specified as "variants". The keyword "Variants:" announces the fact that a test case has variants. In the example

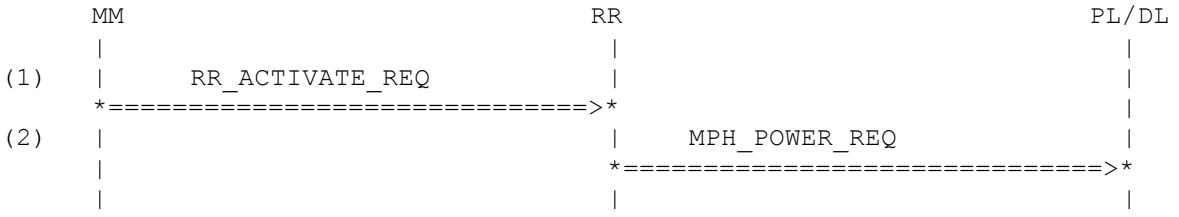
Variants: <A>...<E>

The currently defined test case has 5 variants. The last character of the test case name will take on the letters "A" through "E". The usage of Variants is optional. Variants may be specified before the preamble(s).
If a test uses different preambles, the variants are noted for each variant as in the following example

Preamble: <A>RR022
<B>RR027A

4.3.3.5 Primitive Sequence Chart

Immediately following the keyword "Preamble" or "Variants" a Primitive Sequence Chart of the Module Test Case test case is specified. Here is an example of such a chart which is part of a Module Test for the GSM entity RR:



The Entity Under Test (EUT) is RR. The other entities affected in this test case are MM and PL/DL. Two Primitives are sent in this test case: RR_ACTIVATE_REQ and MPH_POWER_REQ. The direction of the primitive flow is indicated by the arrows (=====>). In this example, the Test Application Process (TAP) first sends a RR_ACTIVATE_REQ to the EUT and then waits for a MPH_POWER_REQ, which must be sent by the EUT.

The strings indicating the entities are informative only and have no significance for later processing of the test document. There must be always three columns. The first and the last line of the chart must be empty.

Each primitive is to describe by a mandatory primitive name and the number of primitive (first line), an optional comment in the next line (e. g. the name of the message the primitive carries) and the mandatory arrow (next line). In every line there must not be parts of the description of different primitives. The name of primitives must be identical to names of primitives in the parametrization section of the test case (see section 3.3.3.6)

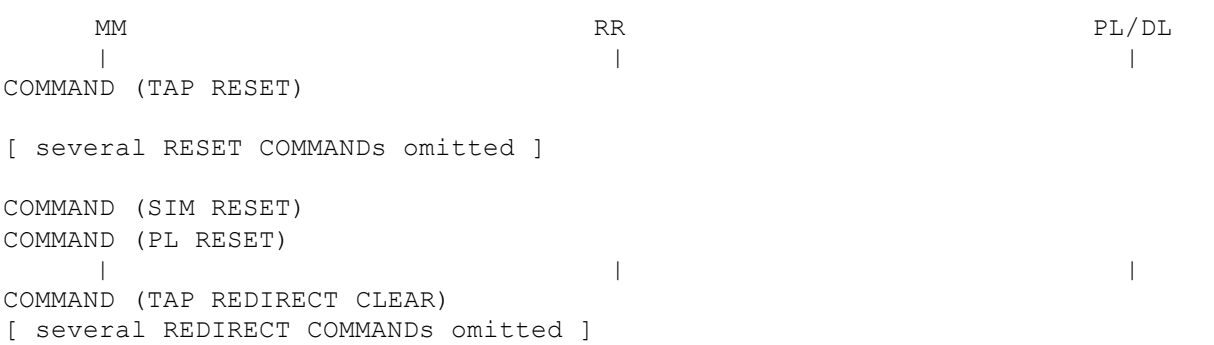
Before, between or after the primitives (or instead of them) there may be inserted lines

- COMMAND
  (description see later in this section)
- TIMEOUT (time)
  where time is a non negative integer. The time is measured in msec. The TAP is suspended for the given time. If placed before the expected receiving of a primitive (i.e. in the example above the MPH_POWER_REQ – the arrow is directed from the EUT), the instruction adds time to the default waiting time (10 s) for a primitive. A test will fail if EUT sends no primitive during the default waiting time plus time.
- MUTE (time)
  where time is a non-negative integer. The time is measured in msec. A test will fail if the entity under test sends a primitive during the given time. The instruction may be used to check whether a timer is stopped or if there are unexpected primitives sent from EUT.
- START_TIMEOUT (time)
  where time is a non negative integer. The time is measured in msec. The TAP start internally a timer with for the given time. It is not waited for the expiration of the timer. Subsequently all usual actions i.e. sending and receiving of primitives can be performed. With WAIT_TIMEOUT it is possible to wait for expiration of the started timer.
- WAIT_TIMEOUT
  waiting for the expiration of the timer started by START_TIMEOUT. This command now acts like MUTE for the remaining time of the timer. If the timer was expired before WAIT_TIMEOUT was called, the test will fail.
- REPEAT (var,cnt)
  define a repeat loop. The parameter var is the C name of the loop index. It must differ from all TDS_HANDLE values used inside of the loop. With cnt the number of times the loop is to be executed is given.
- ENDREPEAT
  closes the repeat loop defined with REPEAT before.

All instructions must begin in the first character of a line and require one blank space between the command name and "(".
It is possible to replace the timer values by symbolic constants (e.g. "TIMEOUT (TIMERVALUE1)"). These constants have to be defined in the Parameters Section of the test case document or are known through an included header file. Additionally, the user can specify different timer values for variants of test cases (see section 4.3.3.6 for further details).

Typically a Module Test has at least one primitive sent to the EUT (stimulus) and at least one primitive sent from the EUT to another entity (response).

Primitive sequence charts may contain commands for dynamic configuration purposes. The following chart contains commands to set up the EUT (entity RR). All communication paths are reset. The data flow from and to RR is redirected to the Test Application Process (TAP) (indicated in bold font below). Typically this initialization is part of the first test case, whereas some of the commands are omitted for the reason of clarity.



```
COMMAND (PL REDIRECT CLEAR)
      |                                 |                                 |
COMMAND (MMI REDIRECT MM NULL)
[ several REDIRECT COMMANDs omitted ]
COMMAND (MMI REDIRECT PL NULL)
      |                                 |                                 |
COMMAND (CC REDIRECT MMI NULL)
COMMAND (CC REDIRECT MM NULL)
      |                                 |                                 |
COMMAND (SS REDIRECT MMI NULL)
COMMAND (SS REDIRECT MM NULL)
      |                                 |                                 |
COMMAND (SMS REDIRECT MMI NULL)
COMMAND (SMS REDIRECT MM NULL)
      |                                 |                                 |
COMMAND (MM REDIRECT MMI NULL)
[ several REDIRECT COMMANDs omitted ]
COMMAND (MM REDIRECT DL NULL)
      |                                 |                                 |
COMMAND (RR REDIRECT PL TAP)
COMMAND (RR REDIRECT DL TAP)
COMMAND (RR REDIRECT MM TAP)
COMMAND (RR CONFIG NO_SYS_TIME)
      |                                 |                                 |
COMMAND (DL REDIRECT RR NULL)
COMMAND (DL REDIRECT MM NULL)
COMMAND (DL REDIRECT PL NULL)
      |                                 |                                 |
COMMAND (PL REDIRECT RR NULL)
COMMAND (PL REDIRECT DL NULL)
COMMAND (PL REDIRECT MMI NULL)
      |                                 |                                 |
COMMAND (SIM REDIRECT MM NULL)
      |                                 |                                 |
COMMAND (TAP REDIRECT TAP RR)
      |                                 |                                 |
```

If the test document contains test cases with several stack entities as EUT, then the last command

COMMAND (TAP REDIRECT TAP <EUT>)

must be replaced by several commands of the format

COMMAND (TAP REDIRECT TAP **bbbbbb******** <ENT>)

In this line, bbbbbb is the binary coding of the Service Access Point identifier (each b is one binary digit 0 or 1) and <ENT> is the entity providing the Service Access Point. The tested entities use a line for every Service Access Point.

Example:

COMMAND (TAP REDIRECT TAP **001011******** RR)

In the example, it is assumed that RR has the Service Access Point 11/0x0B.

With a COMMAND it can also be controlled how long the Tap waits for the arrival of a primitive. The default value is 10 sec. With

COMMAND (TAP TIMEOUT 6000)

it can e.g. set to 6 sec. The value following the keyword TIMEOUT is interpreted to be the timeout measured in milliseconds.

Another application for Tap internal COMMANDs is the possibility to switch the parking[1] behavior off and on (COMMAND (TAP PARKING OFF) resp. COMMAND (TAP PARKING ON)).

[1] If a primitive is received that is not expected, the usual behavior of the Tap is to "park" that primitive. I.e. it is stored. Later if it is expected it will be considered. It is an error if at the end of a test case there are still parked primitives. If a strict sequence of primitives is required it is useful to switch the parking behavior off.

4.3.3.6 Parametrization

The Message Sequence Chart is followed by a section describing the values of timer statements and of the Primitives sent to and received from the Entity Under Test. This section starts at the keyword "Parametrization". Here is an example for a Parametrization section:

Parametrization

Timer Parameter		Value
TIMERVALUE1	<A>	500
	<B>	1000
TIMERVALUE2		1500

Primitive	Parameter	Value
(1) RR_ACTIVATE_REQ		
	plmn	PLMN_ID_123
	op	OP_MODE_TEST_SIM
	cksn	CKSN_NOT_PRES
	kcv	KCV_12345678
	accc	ACC_CTRL_CLASS_0008
	imsi	MOBILE_ID_IMSI_HPLMN
	tmsi	MOBILE_ID_TMSI
	thplmn	TIME_HPLMN_VALID
	bcch_info	BCCH_INFO_EMPTY
(2) MPH_POWER_REQ		
	pch_interrupt	PCH_INTERRUPT
	freq_bands	NOT_USED

In the Primitive Sequence Chart timer values can be specify directly (e.g. "TIMEOUT (500)") or by symbolic constants (e.g. "TIMEOUT (TIMERVALUE1)"). It is possible to define these constants in included (header) files, in the Parameters Section of the test document or in the Parametrization of a test case. For each timer constant, which is not defined elsewhere there has to be a definition in the Parametrization. The symbolic constants should be unique within a test document. After the keywords "Timer Parameter" and "Value" the definition of the timer parameters starts: The timer parameters (in the example TIMERVALUE1 and TIMERVALUE2) in the first column are followed by non-negative integer values (timer values are measured in msec). If it is necessary to have different timer values in variants of a test case the value must be prefixed with the letter of the variant (as it is shown for TIMERVALUE1).
If all timer parameters have been specified the parametrization of the primitives follows.

For each primitive in the Primitive Sequence Chart there must be one description of the primitive parameter in the Parametrization section. Every primitive parameter (e.g. "plmn") occurring in the primitive description of the Service Access Point document must occur in the same order in a line of the parameter section after the appropriate primitive name. This is a list of acceptable values (e. g. PLMN_ID_123):

- If the description of the parameter in the Service Access Point document contains a value table, any symbol in the c-macro column may be used.
- If the primitive parameter is of type UBYTE, USHORT or ULONG, then an appropriate <TDS_NAME> of a BYTE, SHORT or LONG macro from the parameter section of test document may be used.
- If the primitive parameter is of type array of UBYTE, then an appropriate <TDS_HANDLE> of a BEGINARRAY[_PART] macro from the parameter section of test document is to be used.
- If the primitive parameter is of type STRUCT, then an appropriate <TDS_HANDLE> of a BEGIN_PSTRUCT macro from the parameter section of test document is to be used.
- If the primitive parameter is of type array of USHORT, then an appropriate <TDS_HANDLE> of a BEGIN_SHORT_ARRAY[_PART] macro from the parameter section of test document is to be used.

- If the primitive parameter is of type array of ULONG, then an appropriate <TDS_HANDLE> of a BEGIN_LONG_ARRAY[_PART] macro from the parameter section of test document is to be used.
- If the primitive parameter is of type array of STRUCT, then an appropriate <TDS_HANDLE> of a BEGIN_STRUCT_ARRAY[_PART] macro from the parameter section of test document is to be used.
- If the primitive parameter is a bit field longer than 24 bits then an appropriate <TDS_HANDLE> of a SET_BITBUF macro from the parameter section of test document is to be used.

If the value of a parameter differs for variants of the test case, then for each variant a line must be coded which is prefixed with the letter of the variant, e. g.

```
<A>          accc    ACC_CTRL_CLASS_0008
<B>          accc    ACC_CTRL_CLASS_0004
```

Constant values (magic numbers) are not allowed in the "Values" column of the parameter description.

If the primitive parameter contains a message, the keyword SDU is used to start the definition of the message. The information elements of the message are surrounded by braces (SDU {...}). If the message contains a high layer message, the keyword hl_sdu is used to start the definition of the nested message. The information elements are again surrounded by braces (hl_sdu{...}).

- The first line inside the braces describes with the parameter name component, the entity to which the message belongs.
- The second line notes after the parameters name the direction of the message by the keywords UPLINK or DOWNLINK respectively. For messages defined for both directions the keyword BOTH is available here.
- The third line notes with the parameter name pd, the message type of the message. The value is the short name of the appropriate messages from the message catalog in upper letters.
- The fourth line uses as parameter name md for ABIS messages and ti for all other messages. The value refers to a BYTE macro with the coding of the message discriminator for ABIS messages or the coding of skip indicator or transaction identifier for other messages. If the feature of an extended transaction identifier shall be used, it is possible to a further parameter tie on the fifth line following the ti.

The following lines contain the parameters of the message catalog after the msg_type (in the same order). There is no line with the parameter name msg_type in the test document.

For the lines inside the braces (the information elements) of the message, the previous paragraph is valid (by replacing "Service Access Point" by "Air Message" document and "primitive parameter" by "information element. Here is an example of a primitive containing a system information message:

```
(1) MPH_UNITDATA_IND
                                arfcn           ARFCN_43
                                sdu
                                {
                                component       RR
                                direction       DOWNLINK
                                pd              D_SYS_INFO_1
                                ti              TI_0
                                cell_chan_desc  CELL_CHAN_DESC_1
                                rach_ctrl       RACH_CTRL_1
                                }
```

To build erroneous messages, use the name of a SET_BITFIELD macro in the same line instead of the information elements surrounded by braces in the following lines. In the referenced byte array, the entire erroneous message is to be coded. BEGIN_ARRAY or FIELD macro are also possible, but platform dependent and therefore obsolete.

There exist the special parameter values NOT_USED, DISPLAY_ONLY and FORBID. The value NOT_USED for a sent parameter has the meaning that the parameter is ignored, i.e. nothing is filled into the primitive/message. NOT_USED for a received parameter means that the received data is not

checked. A value of DISPLAY_ONLY means that the received data is shown in the traces but not checked against any given data. With FORBID it is checked that the (probably optional) parameter is not included. If it is included, the test case fails. Both, DISPLAY_ONLY and FORBID, are only valid on receiving.

4.3.3.7 History

The end of a test case definition is indicated by the "History:" section, introduced by the keyword "History":

History:	04.07.97	DL	Initial
	05.05.98	VK	Revised (one prim removed)

An entry in the history section contains three items: a date, the initials of the author who changed the test case and a comment describing the changes.

4.3.4 Suites Section

The section headed "Suites" is supported for component tests in TDL language only. It contains an additional way to order or to reorder test cases with suites. The main intention of suites is the possibility to define repeated loops on the base of test cases. In addition to loops it is possible to select a test case randomly. During the execution of test suites, Preambles are not performed to avoid e.g. unnecessary initialization actions inside of loops.
Each suite is a list of

- test cases and/or
- other suites and/or
- commands for the selection of a certain suite or the repetition of suites.

Preambles inside the test cases are not needed. If they occur, they are ignored.

4.3.4.1 Syntax

The suite section has the following syntax:

<comment> ::= /*<any sequence of characters containing neither /* nor */>*/
<white space> ::= <any sequence of blank, tab and/or new line characters>
note: Comments and white spaces are ignored

<test case name> ::= <name of a test case from the section "test cases">
remember: Such a name consists of the name of the test document, three digits and optionally a letter for a variant as described in section 3.3.3

<suite identifier> ::= <any sequence of letters, digits and underline character>
note: The sequence must have at least one character. Suite identifiers are case sensitive.

<integer> ::= <any sequence of digits>
note: The value of an integer has its normal meaning and it must fit into an integer of the C language.

<suite type> :: <internal> | <able to run>
<internal> ::= INT_
<able to run> ::= SUI_
<full suite name> ::= <suite type><suite identifier>
Note: "able to run" means that the suite may be processed. Its first test case must contain the redirection commands as described in section 3.3.3. "Internal" means that the suite is used to construct other suites in an easily understandable manner for humans.

<member> ::= <test case name> | <full suite name>| <select command> | <repeat command>
<evaluated member> ::= <member> , <integer>

<name list > ::= <member> | <name list> , <member>
<evaluated name list> ::= <evaluated member> | < evaluated name list>, <evaluated member>

<select command> ::= SELECT(< evaluated name list>)
<count> ::= <integer> | EVERY | FOREVER
<repeat command> ::= REPEAT(<count>,<name list >)

<suite definition> ::= <full suite name> : <name list> ;
note: The full suite names in suite definitions must be unique in the suite section.

<suite section> ::= <suite definition> | <suite section><suite definition>
<suite chapter> ::= <suite section> ENDSUITES

4.3.4.2 Processing of suites

All suites of type "able to run" may be processed by interpreting its compound list of members:

- If the member is a test case name, the appropriate test case is performed (without its preamble if the test case contains one). This requires that the test case is compiled to a dll file. After processing all primitives of the test case, this member is complete.
- If the member is a full suite name, the definition of it is processed in the same way as the "calling" suite.
- If the member is a select command (and if this select command is not processed during the processing of a repeat command with the count EVERY),the integers of its evaluated name list are interpreted as probabilities. The integers should be between 1 and 99. The sum should be 100. By a random number scaled to 100, a member of the evaluated list is selected. This member is processed in the same way. After the processing of this member, the processing of the select command is complete.
- If the member is a repeat command and the count is an integer, its name list is processed. If the processing is complete, a counter is incremented. If it has not reached the integer value, the name list is processed again and so on. If the counter reaches the count value, the processing of the repeat command is complete.
- If the member is a repeat command and the count is the value FOREVER, an infinite number of repetitions is started. Note: such a suite may never pass.
- If the member is a repeat command and the count is the value EVERY, the number of repetitions is defined implicitly by the select commands processed during the processing of the name list of the repeat command. An other member of a called select command (if there is one select command called) or an other combination of members of different select commands is selected on every repetition. Such a repeat command is complete if its name list is processed so often that every combination of members of called select commands is processed.
- If a member of a list is complete, the next member of the list is processed if present. Otherwise, the processing of the list is complete.

4.3.4.3 Example

An example of a suite section follows.

```
/* Routing  */
SUI_Init: MCOMP000, MCOMP009;

/* minimal recources */
INT_Res1: MCOMP020, MCOMP021, MCOMP025;
/* middle resources */
INT_Res2: MCOMP300, MCOMP302, MCOMP304;

/* preparing the BTS */
SUI_SYS_INFO1: SUI_Init, INT_Res1, MCOMP040A;
SUI_SYS_INFO2: SUI_Init, INT_Res2, MCOMP040A;

/* Immediate Assign */
INT_ImmAss_LUT: MCOMP060A, MCOMP061A;
```

```
INT_ImmAss_MOC: MCOMP060B, MCOMP061A, MCOMP080;
INT_ImmAss_MTC: MCOMP060C, MCOMP061A, MCOMP062A;

/* Location UPDATE without and with TMSI reallocation */
INT_LU_WO_Realloc: MCOMP062A, MCOMP066;
INT_LU_With_Realloc: MCOMP062A, MCOMP068, MCOMP070A;
INT_LU_TMSI_delete: MCOMP062A, MCOMP069A, MCOMP070A;

/* Location update variants */
INT_LU: SELECT( INT_LU_WO_Realloc, 30, INT_LU_With_Realloc,60,
INT_LU_TMSI_delete, 10);

/* Release of SDCCH */
INT_Rel_SDCCH_OK: MCOMP130A, MCOMP131A;
INT_Rel_SDCCH_ERR: MCOMP130A, MCOMP137;
INT_Rel_SDCCH: SELECT(INT_Rel_SDCCH_OK,98, INT_Rel_SDCCH_ERR,2);

/* Complete Location update */
INT_LU_EVERY: REPEAT(EVERY,INT_ImmAss_LUT, INT_LU, INT_Rel_SDCCH);
INT_Comp_LU: INT_ImmAss_LUT, INT_LU, INT_Rel_SDCCH;

/* Suites for Location Update */
SUI_LU_0:  SUI_SYS_INFO1, INT_Comp_LU;
SUI_LU_1:  SUI_SYS_INFO1, INT_LU_EVERY;
SUI_LU_2:  SUI_SYS_INFO2, INT_LU_EVERY;
SUI_LU_3: SUI_SYS_INFO1, REPEAT(10, INT_Comp_LU)

ENDSUITES
```

The end of the suite section is indicated by the keyword ENDSUITES.
In this example, the processing of the suite SUI_LU_0 calls the test cases

SUI_SYS_INFO1	SUI_Init	**MCOMP000, MCOMP009**	
	INT_Res1	**MCOMP020, MCOMP021, MCOMP025**	
	MCOMP040A		
INT_Comp_LU	INT_ImmAss_LUT	**, MCOMP061A**	
	INT_LU	SELECT(INT_LU_WO_Realloc, 30, INT_LU_With_Realloc, 60, INT_LU_TMSI_delete, 10) assume a generated random value between 30 and 89	**MCOMP062A, MCOMP068, MCOMP070A**
	INT_Rel_SDCCH	SELECT(INT_Rel_SDCCH_OK,98, INT_Rel_SDCCH_ERR,2) assume a generated random value between 0 and 98	**MCOMP130A, MCOMP131A**

The processing of SUI_LU_1 processes SUI_SYS_INFO1 and 6 repetitions of the sequence of

- INT_ImmAssLUT,
- one of three variants from INT_LU and
- one of two variants of INT_Rel_SDCCH:

4.4 Multi Layer Test Cases

4.4.1 Purpose

Multi Layer Test Cases are implementations of the GSM conformance tests according to the GSM 11.10 specification. The Test Case Script Language (TCSL) is used to define Multi Layer Test Cases.

This language is described in a separate document (ref. [C_6147.302]). The structure of test case documents containing Multi Layer Test Cases is similar to the structure of a document containing Module Test Cases. At top level there is a section "Parameter" and a section "Test Cases". However the contents are different.

4.4.2 Parameter Section

The Parameter Section of a test case document containing Multi Layer Tests defines test data which is used in the section "Test Cases". TCSL provides the following constructs to define test data:

- IE_BEGIN start of the definition of an Information Element
- BF definition of a Bit Field
- IE_END end of the definition of an Information Element

- MSG3_BEGIN start of the definition of a Layer 3 Message
- IE reference to an Information Element
- MSG3_END end of the definition of a Layer 3 Message

These language elements are described in a separate document (ref. [C_6147.302]).
The definition of test data in a Multi Layer Test Case is done in two steps:

- definition of Information Elements (using IE_BEGIN, BF, and IE_END)
- definition of messages (using MSG3_BEGIN, IE, and MSG3_END)

This two step way of defining test data items reflects the way the air interface messages are defined in the GSM specification.

4.4.3 Test Case Section

The section headed "Test Cases" contains groups of test case definitions. A test case group contains any number of test cases. Each test case is defined in a separate section which has the following structure:

- Heading (Level 3) ref 4.3.3.1 Heading
- Description ref 4.3.3.2 Description
- Preamble ref 4.3.3.3 Preamble
- Variants ref 4.3.3.4 Variants
- Script
- History ref 4.3.3.7 History

The sections Heading, Description, Preamble, Variants and History are the same for both Module Tests and Multi Layer Tests. Refer to the referenced section of the Module Test Cases. There is no "Primitive Sequence Charts" or "Parametrization" section in Multi Layer Test specifications. Instead, a section named "Script" is included in the Multi Layer Test. This section contains the specification of the exchange of Layer 3 Air Interface Messages. Here is an example from a Multi Layer Test case:

Script:

```
ISS_INIT (4);

BS_SET_SYS_INFO ( 0 , system_information_type_1 );
BS_SET_SYS_INFO ( 0 , system_information_type_2 );
BS_SET_SYS_INFO ( 0 , system_information_type_3 );
BS_SET_SYS_INFO ( 0 , system_information_type_4 );
```

```
BS_SET_SYS_INFO_SACCH ( 0 , system_information_type_5 );
BS_SET_SYS_INFO_SACCH ( 0 , system_information_type_6 );

BS_SET_SCH ( 0 , BSIC , RFN );
BS_SET_ARFCN ( 0 , ARFCN_BCCH );
BS_SET_POWER ( 0 , -50 );
BS_ON_OFF  ( 0 , TRUE );

COMMAND ("MMI CONFIG KEY_SEQUENCE=<#*91*1#>");   /* Power On          */

ISS_DELAY (20000);

COMMAND ("MMI CONFIG KEY_SEQUENCE=<030390941l7>");/* Dial              */

BS_RACH_AWAIT(0,channel_request_moc,SILENT);
BS_CONFIG_CHANNEL (0, AGCH, UNACK, SAPI_0);
BS_STORE_RACH_PARAMS (0, 0);
BS_MSG3_SEND (0,immediate_assignment,SILENT);

BS_CONFIG_CHANNEL (0, SDCCH, 1, SAPI_0);
BS_MSG3_AWAIT(0,cm_service_request,SILENT);
```

The following steps are performed in this example:

- initialization of the system
- Power On sequence
- Delay for network and cell selection
- MOC up to the cm_service_request message

The language elements that may be used to specify the Script part of a Multi Layer Test case are described in a separate document (ref. [C_6147.302]). However, this is a brief list of the most important TCSL language elements used in the Script part:

ISS_INIT	(NUM_BS)	initialize ISS
ISS_DELAY	(MILLI_SEC)	delay in milliseconds
BS_SET_SYS_INFO(BS_IDX,SI)		define system information messages
BS_ON_OFF	(BS_IDX,ON_OFF)	switch on a base station
BS_MSG3_SEND	(BS_IDX,MSG3,COM)	send a Layer 3 message
BS_MSG3_AWAIT	(BS_IDX,MSG3,COM)	receive and compare a Layer 3 message

5 Test Case Creation

The following steps must be carried out to create an executable test case:

- apply the Test Case generator TdsGen
- use the tool MkTc to compile and link the test case

5.1 Test case generator TdsGen

The test case generator processes the Test Case definition document (ref [3 Test Case Definition]) and produces a data definition script (*.def), a test definition script (*.tds) for each test case definition. All files are stored in %TDSDIR%\<Entity Name>. The environment variable TDSDIR must be active when the test case generator processes the test case definition document. The Entity Name is actually the name of the test case definition document. An example for a test definition script is %TDSDIR%\RR\RR027.TDS. If RR027 had variants (ref [4.3.3.4 Variants]) A..E, then five files RR027A.TDS through RR027E.TDS would have been created. As an example here is the output of the test definition script related to variant A of the test case RR001.

```
#include "MACROS.H"
#include "RR.DEF"
```

```
TESTCASE (RR001A)

#include "RR000.TDS"

/*
 TITLE:  Start cell selection

 DESCRIPTION:  A cell selection without BCCH information is started. Variant A: no SIM is inserted. Variant B: SIM
is inserted, test SIM card, IMSI in the HPLMN, TMSI Variant C: SIM is inserted, normal SIM card, IMSI in the HPLMN,
no TMSI Variant D: SIM is inserted, test SIM card, IMSI in the HPLMN, no TMSI Variant E: SIM is inserted, normal
SIM card, IMSI in the HPLMN, TMSI
*/

SEND (RR_ACTIVATE_REQ)
SET_PARAM ("plmn", PLMN_ID_EMPTY)
SET_PARAM ("op", OP_MODE_EMPTY)
SET_PARAM ("cksn", CKSN_NOT_PRES)
SET_PARAM ("kcv", KCV_EMPTY)
SET_PARAM ("accc", ACC_CTRL_CLASS_0000)
SET_PARAM ("imsi", MOBILE_ID_NOT_SET)
SET_PARAM ("tmsi", MOBILE_ID_NOT_SET)
SET_PARAM ("thplmn", TIME_HPLMN_EMPTY)
SET_PARAM ("bcch_info", BCCH_INFO_EMPTY)
ENDSEND

AWAIT (MPH_POWER_REQ)
COMP_PARAM ("pch_interrupt", PCH_INTERRUPT)
SKIP_PARAM ("freq_bands")
ENDWAIT

ENDCASE
```

Each test definition script includes the file MACROS.H. This file contains the definitions of the macros used to define test data (e.g. BEGINARRAY) and macros generated by the test case generator (e.g. SEND and AWAIT). The test data defined in the test case definition document is output to the file <Entity Name>.DEF, for example RR.DEF. The test case file which is used as a Preamble (ref [4.3.3.3 Preamble]) is included at the top of the test case (#include "RR000.TDS" in the example).

In the case of an error, the user is briefly informed by a dialog box. A detailed description of the error is placed into the files <Entity Name>.err and <Entity Name>.log (e.g. RR.ERR and RR.LOG). Besides, a file named <Entity Name>.war is created to store warnings which TdsGen detects.

There are two ways to start the test case generator, either from within WinWord or from the command line.

5.1.1 TdsGen (WinWord)

The test case generator may be called at any time for the currently edited test case definition document. The WinWord menu options <WCS_MACROS> <Generate tds> must be used. If the menu option <WCS_MACROS> is missing, then most likely the DOT file (TUCT_TECH) is missing (ref [4.1 Creating and editing test case documents]).

5.1.2 TdsGen (Command Line)

To call the test case generator without using WinWord, the scripts documented in the sections 5.2 and 5.3 must be used.

5.1.3 Generating suite file

The suite section of a component test document is used to offer an additional way to control the order of test cases. To use this option, the suite section must extracted from test document and stored as a plain file. The name of this file is <Entity>.sui. There are two directories where the file is stored: first, in the directory where TdsGen has produced the tds-files (controlled by the environment variable (%TDSDIR%). This location is used for tdscheck (see **Error! Reference source not found.**) processing. The second location of the suite file is the same directory where the test case DLLs are generated. The location of this directory is dependent on the project for that the test cases are com-

piled. The name of this directory is printed by the test case generation scripts explained in the following sections. Those scripts also perform the extraction of the suite file.

5.2 Generating executable test cases: MKTC

After the test case generator has created the test definition scripts the Microsoft C-Compiler and Linker is used to compile and link the scripts and to generate executable test cases. These executable test cases are Dynamic Link Libraries. The tool MkTc is provided to make this process easy. In order to create an executable test case, the following line has to be executed from a 4NT box:

MKTC [-gen] [-chk] [-i <idir>] [-td <tdir>] [-eut <eutname>] <Entity> <TestCase> [<TestConfig>]
where <Entity> is the name of the EUT (e.g. RR) resp. the name of the test document
<TestCase> denotes the Testcase (e.g. 012A – in the form RR012A it will also be accepted)
<TestConfig> may be used optionally if the name of the test configuration differs from the name of the EUT (e.g. in the case of the MMI Framework, TestConfig is MFW). It is also possible to submit directly the name of the EUT in the form EUT=<eutname>.

Examples: in order to create an executable test case for the test case RR012A, the test definition script RR012A.TDS has to be compiled and linked. For BMISS200 the script BMISS200 is used and the test configuration MFW has to be submitted.

MKTC RR RR012A
Or: MKTC BMIT 200 MFW
Or: MKTC BMIT 200 EUT=ACI

The output files are RR012A.DLL and BMIT200.DLL respectively. The output directory depends on protocol stack to be tested. MKTC reports, where the file is placed. If an error occurs, a detailed error description of the compiler and linker is output to the file <Test case Name>.err (e.g. RR012A.ERR, BMIT200).

Options: -gen: call tdsGen before generating the DLL
-chk: call tdsGen and afterwards the test case checker tdscheck (see section 5.4) before generating the DLL
-i <idir>: set <idir> as additional include directory for test case compilation
-td <tdir>: set <tdir> as directory for the test cases
-eut: same as EUT=<eutname> as last parameter

5.3 Generating all executable test cases for a test case document: MKALLTC

The batch file MKALLTC.BAT is provided to perform the preparing of all test cases of a test document with one command. It first calls TdsGen.exe via the WinWord macro macGen and then generates for each existing .tds file the corresponding DLL containing the executable test case. Corresponding to MKTC.BAT it is also possible to submit a different test configuration. The command is called from a 4NT box:

MKALLTC [-chk] [-co] [-to] [-f] [-r [<first>]-[<last>]] [-k] [-i <idir>] [-td <tdir>] [-eut <eutname>]
<Entity> [<TestConfig>]

Examples:
MKALLTC RR

Or: MKALLTC BMIT MFW
Or: MKALLTC BMIT EUT=ACI

All generated executable test cases are placed in a directory whose name is reported by MKALLTC.BAT.

Options: -chk: call the test case checker tdscheck (see section 5.4**Error! Reference source not found.**) before generating the DLLs.
-co: compile only - generate TDS files only; don't compile test cases
-to: generate TDS files only; don't compile test cases
-f: generate final test only (test, that are not preambles)
-r: generate ranges of the available test cases from <first> to <last>
-k: keep (don't delete) older testcases (useful when using -r)
-i: set <idir> as additional include directory for test case compilation
-td: set <tdir> as directory for the test cases
-eut: same as EUT=<eutname> as last parameter

5.4 TDSCheck

Test documents may contain errors which are not detected by TDSGen and the C compiler. The result is a failure of test cases which could be detected before compiling time. Examples are misspelled parameter names, wrong order of parameters in primitives/messages and references to parameter values, which are not or incorrectly defined.

TDSCheck parses all files produced by TDSGen (*.def, *.tds and *.sui, if present) to detect all problems in a test document which may be found without running the test cases, e.g. formal and logical errors or variations in the structure of types defined in SAP or Message Documents and used in the test document. The program produces a common list of warnings and error messages for all those files. After detecting an error parsing is continued as soon as possible to detect further problems. The diagnostics of TDSCheck are written to a file named <Entity Name>.chk which is placed in the same directory as the parsed files. Before terminating TDSCheck calculates some statistics (number of errors, …) for the current test document and writes the results to the file specified above.

The user may call TDSCheck directly or from test generation scripts (MKTC, MKALLTC, …) with the parameter "-chk".
In the former case the program is called with zero, one or two parameters from a 4NT box:

tdscheck [<path> [<test document>]]
where <path> is the path to the files to parse
<test document> is the name of the test document which defines the test cases

Example:
tdscheck \GSM\Condat\MS\TDS\mm MM

It is also possible to specify these parameters in the initialization file but if given at command line the parameters in the ini file are ignored.

5.4.1 The initialization file

After the call of TDSCheck the program tries to read the file tdscheck.ini located in the same directory as the program. This file contains options for the program. A default initialization file with the latest options is provided by the developers each time they release a new version of TDSCheck. It is

checked in as tdscheck_orig.ini in the GPF binary directory. The user has to save this file as tdscheck.ini (DO NOT FORGET THAT YOU OVERWRITE YOUR OLD tdscheck.ini WITH THIS) and may change this file for his purposes.

Within the initialization file it is possible to use comments like in standard C. The options can appear in any order (except the "IDENTICAL"s, see below). Currently the following options are defined.

Name Explanation and example

PATH path to files to parse, ignored if first parameter is given at command line
Example: PATH = \GSM\Condat\MS\TDS\ACISAT

ENTITY name of the test document which defines the test cases, first part of the *.def file,
ignored if second parameter is given at command line
Example: ENTITY = ACISAT

CDGINC path to files *.val, mconst.cdg and pconst.cdg, which define constants, primitives, messages, …
ignored if environment variable CDGINC or CDGINCDIR can be found
Example: CDGINC = \GSM\Condat\MS\CDGINC

VAL_FILES Path and file name containing the list of *.val files included in the test cases, if the file is not found or the file does not contain the format this file is expected to have,
e.g. #ifdef TEST_ENTITY_<XX>
#include "yyy.val"
....
#endif ,
TDSCheck reads all *.val files from CDGINC
if this happens TDSCheck is not able to find errors which arise from more than one definition
Example: VAL_FILES = \GSM\Condat\MS\TDS\entity_ccd.h

The following examples contain the recommended values for the options (as given in the default initialization file).

ONLY_TO_FILE The errors and warnings detected by TDSCheck are written only to the * ~.chk file (no display output)
Possible Values: "YES" or "NO"
ONLY_TO_FILE = YES

MAXERRORS maximum number of errors to be displayed
possible values: number or "NO" for no limit
Example: MAXERRORS = NO

WARNINGS Warnings shall or shall not be displayed.
Possible values: "YES" or "NO"
Example: WARNINGS = YES

MAXWARN The maximal number of warnings to be displayed,
only valid if WARNINGS = YES
Possible values: number or "NO" for no limit
Example: MAXWARN = NO

OBSOLETE Warnings about obsolete constructions (e.g. FIELD) shall or shall not be displayed,
only valid if WARNINGS = YES

Possible values: "YES" or "NO"
Example: OBSOLETE = NO

IDENT_REDEF Warnings about identical redefinitions shall or shall not be displayed,
only valid if WARNINGS = YES
Possible values: "YES" or "NO"
Example: IDENT_REDEF = NO

REPEAT_MSG Repeated messages for handle names which are not properly defined
Possible Values: "YES" or "NO"
Example: REPEAT_MSG = YES

OPT_ERR A warning about a skipped parameter which is not optional shall or shall not be displayed,
only valid if WARNINGS = YES
Possible values: "YES" or "NO"
Example: OPT_ERR = NO

OLD_TAP The program shall not or shall accept forward references of handles of structure definitions,
forward reference means using a handle X inside another BEGIN_xSTRUCT or inside an BEGIN_(P)STRUCT_ARRAY before defining BEGIN_xSTRUCT("...", X) itself
select YES if you don't use tap2_xxx.exe
Possible values: "YES" or "NO"
Example: OLD_TAP = NO

WARN_OLD The program shall or shall not display a warning if it finds *.tds file which is older than the respective *.def file, independent of the option such a file isn't parsed,
only valid if WARNINGS = YES
Possible Values: "YES" or "NO"
Example: WARN_OLD = YES

MISSING_PAR The program shall or shall not display a warning if it doesn't find instructions concerning optional parameters before the end of sdu or hl_sdu, select YES if you want to perform tests and check their results for parameters ommitted in test document, but set by tested entity
Note: Missing MANDATORY parameters before the end of sdu or hl_sdu and ALL missing parameters before the end of a primitive or the end of a structure definition always cause an error message
only valid if WARNINGS = YES
Possible values: "YES" or "NO"
Example: MISSING_PAR = YES

ACCEPT_VAL The program shall or shall not accept numerical values. It shall not or shall display a warning if it finds a decimal or hexadecimal value as value of a SET_COMP instruction or of a parameter inside a primitive instead a name of the the value,
only valid if WARNINGS = YES
Possible values: "YES" or "NO"
Example: ACCEPT_VAL = YES

FIX_LEN The program shall or shall not display a warning if it detects an array of fixed length which is defined with less entries than the fixed length,
select "YES" if you want to be sure, that all array data sent by tested entity are compared,
only valid if WARNINGS = YES
Possible values: "YES" or "NO"
Example: FIX_LEN = NO

ACCEPT_RES The program shall or shall not accept skipped mandatory components named "reserved" (shall not display or shall display a warning),
only valid if WARNINGS = YES
Possible values: "YES" or "NO"
Example: ACCEPT_RES = YES

ACCEPT_ANY The program shall or shall not accept a usage of macro BEGIN_ARRAY or FIELD if the parameter requires BEGIN_SHORT_ARRAY, BEGIN_LONG_ARRAY, SET_BITBUF or BEGIN_ARRAY instead (shall generate a warning only or shall generate an error.
Note: Independent of your choice the program will not check the proper length of generated data in such a case.
only valid if WARNINGS = YES
Possible values: "YES" or "NO"
Example: ACCEPT_ANY = NO

ACCEPT_2COMP The program shall or shall not accept skipped mandatory components with <name> which are preceded by a instruction SET_COMP(c_<name>,...) or SET_COMP(v_<name>,...) and setting the value 0.
Note: If such constructions occur in a test document then this results from a malformed SAP or message catalogue document in almost all cases
only valid if WARNINGS = YES
Possible values: "YES" or "NO"
Example: ACCEPT_2COMP = NO

WRAP The program shall or shall not insert new line characters to try to limit error messages and warnings to line length, select "NO" if you prefer a listing with only 2 lines per error.
Possible values: "YES" or "NO"
Example: WRAP = YES

PRESS_KEY The program shall be finished only after pressing a key or finish without pressing a key, select "YES" if you like to start TDSCheck by double click
Possible values: "YES" or "NO"
Example: PRESS_KEY = NO

END_CHAINS The program shall or shall not display a list of test cases, which have a preamble but are not used as preamble (final tests). If a test case isn't contained in this list it is performed by another test case (contained in the list) or it may be used in suites only.
Possible Values: "YES" or "NO"
Example: END_CHAINS = YES

UNUSED_NAMES Names, which are not used, shall or shall not be displayed
Possible values:"YES" or "NO"
Example: UNUSED_NAMES = YES

SECTION The Developers of TDSCheck have implemented a list of synonyms for each project. These terms with identical meanings are stored after the instruction "IDENTICAL". Each project has its own section starting with "IDENT_FOR". The instruction "SECTION" defines which section shall be evaluated, i.e. which project is the owner of the test document that is currently processed.
Possible values: GPRS, WARP, (UMTS)
Example: SECTION = GPRS

IDENT_FOR Start of the "IDENTICAL"s for each project. If the name given in the "SECTION" instruction and the "IDENT_FOR" instruction are the same the following "IDENTICAL"s will be considered as synonyms for the named project up to the next "IDENT_FOR" instruction (or end of file).

Possible values: GPRS, WARP, (UMTS)
Example: IDENT_FOR = GPRS

IDENTICAL This instruction defines names which have the same meanings in the PS or in the test environment, e.g. structures will be assumed as identical. The developers try to keep the list given in the default initialization file up-to-date.
Possible values: names with the same meaning used in PS development or in the test environment
Example: IDENTICAL = mob_id, mob_id_2

The following example shows how to use the options "SECTION", "IDENT_FOR" and "IDENTICAL". Apart from other options these instructions have to appear in the same order as they are mentioned here.

```
SECTION = GPRS
IDENT_FOR = WARP
  IDENTICAL = e1_id, e1_id_new, e1_id_old
  IDENTICAL = mob_id, mob_id_2
  IDENTICAL = bearer_cap, bearer_cap_2, bearer_cap1, bearer_cap2
IDENT_FOR = GPRS
  IDENTICAL = mob_id, mob_id_2
  IDENTICAL = tmsi, imsi
  IDENTICAL = chan_desc, chan_desc_2, chan_desc_before, chan_desc_before_2, chan_desc_after,
chan_desc_after_2
```

Here the section following after "IDENT_FOR = GPRS" will be evaluated. The structures "mob_id" and "mob_id_2" are regarded as identical.

6 Test Case Execution

A Test DLL is executed in an environment comprised of the following subsystems:

- TAP (Test Application Process) and the Test DLL
- PC implementation of a Condat protocol stack implementation (PS)

The Test DLL contains the executable Test Case and serves as a stimulus for the IUT.
There are two ways to execute test cases:

- start the processes PS and TAP from the command line
- use the tapcaller tool

6.1 Manual start of PS and TAP

In the simplest form, the processes needed for a test are started from the command line of an MS-DOS box within Windows 95 or Windows NT.
To start the PS, type: MS_TI
The name MS_TI may be different depending on the implementation that is to be tested.

To start the TAP, type: TAP2 <windowname> <interface> <entity> <testdir> <testcase>... [T] [H]
The parameter <windowname> denotes the name of the window that TAP2 will open.
The parameter <interface> denotes the interface to be used when stimulating the PS. In the case of a socket interface <interface> is the name of the computer the PS is running on. In the case of a serial interface <interface> consist of the three parts, describing the line: external com port, baud rate, and flow control. Very useful parameters are 1 2 N, which means com port 1, baud rate 38400 bps (yes, really) and no flow control. If a simulated serial interface is used, 0 may be chosen as com port. In this case baud rate and flow control parameter must be omitted.
The parameter <entity> is the name of the entity under test.
The parameter <testdir> is the name of the subdirectory containing the test case.

The parameter <testcase> is the name of the test case. It may be followed by further names of test cases which have to be separated by blanks.
If the optional flag H is set the test runs within hidden windows. It doesn't disturb other work of user. The same effect arrives if the <windowname> NULL is submitted.
If the optional flag T is set the TAP program automatically closes all window of test after termination of last test case. If windows are hidden automatic termination is implied.

For example, in order to execute the test case RLP099 in the directory C:\TMP for a PS communicating via the serial interface, the following command must be issued:

```
TAP2  View  1 2 N  RLP  C:\TMP  RLP099
```

If the serial interface is simulated the following command will deliver the same result:

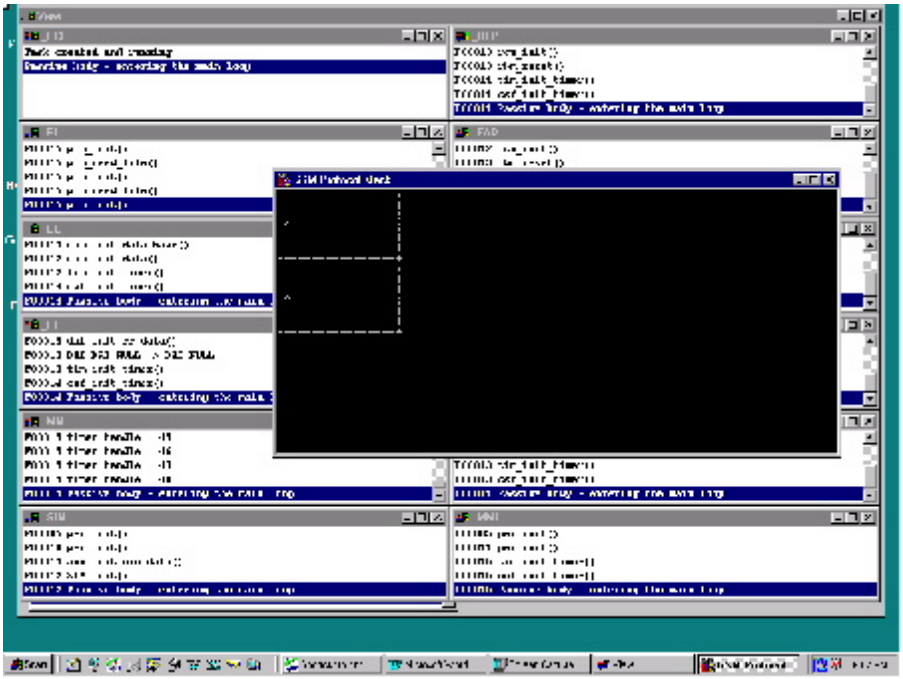
```
TAP2  View  0  RLP  C:\TMP  RLP099
```

The trace files will be stored in the current directory where the TAP is started.
To execute the test case RR022 in the directory D:\DEV\BIN\TEST_RR for a PS running on the computer VK and communicating via the socket interface, the following command must be issued:

```
TAP2  View  VK  RR  D:\DEV\BIN\TEST_RR  RR022
```

If a certain test case is started this requires that the appropriate executable (dll) is compiled and linked successfully. If this test case contains a preamble (which may again contain a preamble and so on) then there is no need (for the test of the case under observation) that the executables for these preambles are up to date or exists. All information about all needed preambles is contained in the executable tested.

After a successful start of a Test DLL the PC monitor screen may look like:



The PC implementation of G23 is running in a window named "G23". The TAP started a window named “View” containing further sub-windows, one window for each entity which has transmitted trace data.

6.1.1 Runalltc

With the batch file runalltc.bat it is possible to start more than one test case with one command. The test cases are executed successively without any necessary user interaction.

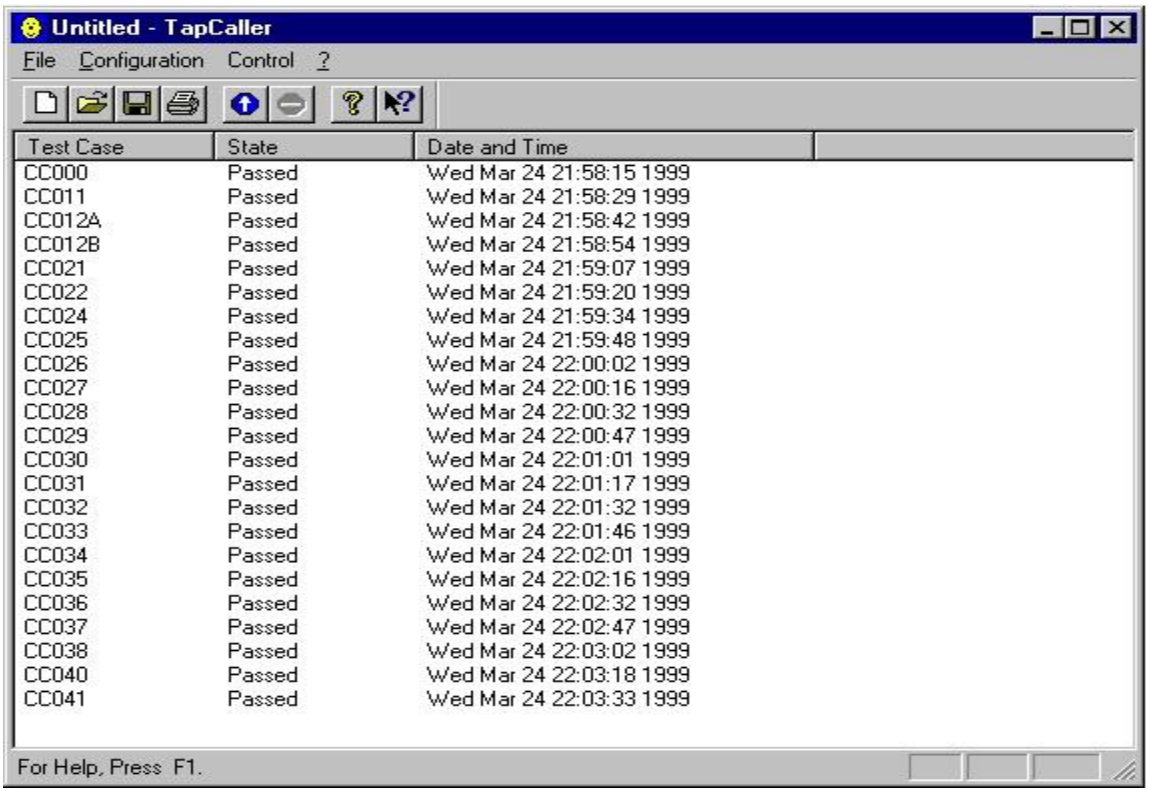
runalltc [-sp] [-st] [-sum] [-p <ps_exe>] [-t <tap_exe] [-r [<first>]-[<last>]] [-d <delay_time>] testname

All previously from testname.doc generated test cases are executed.

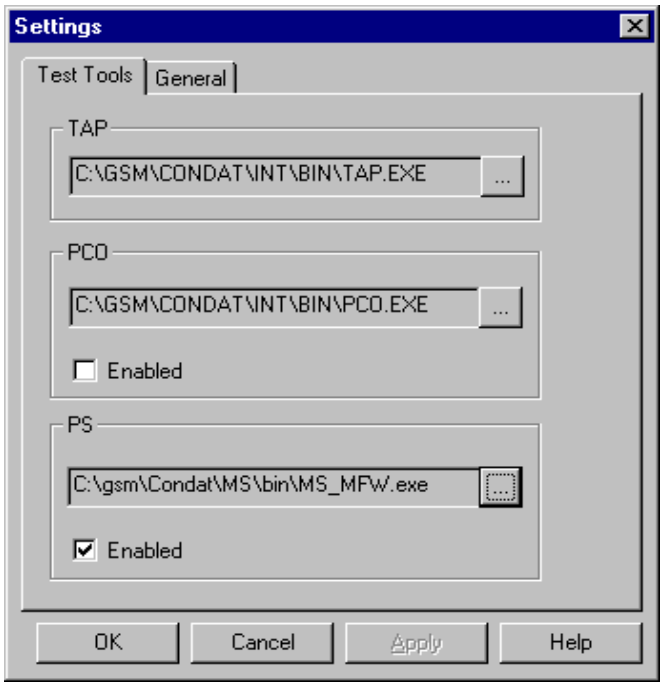
Options: -sp: show window of protocol stack (default: don't show)
-st: show trace windows (default: don't show)
-sum: write summary for each test case in <testname>.txt
-p: take <ps_exe> as protocol stack
-t: take <tap_exe> as tap
-r: run range from <first> to <last>
-d: set <delay_time> (in seconds) after protocal stack starting

6.2 Tapcaller

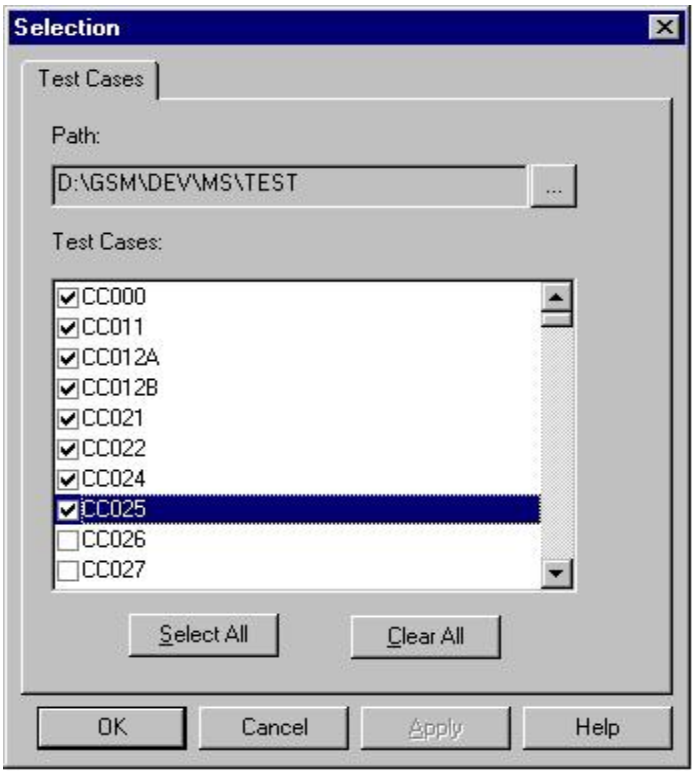
The tapcaller tool is a graphical front-end for executing test cases. After the execution of some test cases, the tapcaller tool may display the following window:



Some setup is necessary to configure the Tapcaller tool. The menu option <Configuration> <Settings> displays the following dialog box. The path to TAP, PS (and PCO if desired) must be entered:



The path of the test cases must be entered and a set of test cases which should be executed may be defined by clicking on the checkboxes:



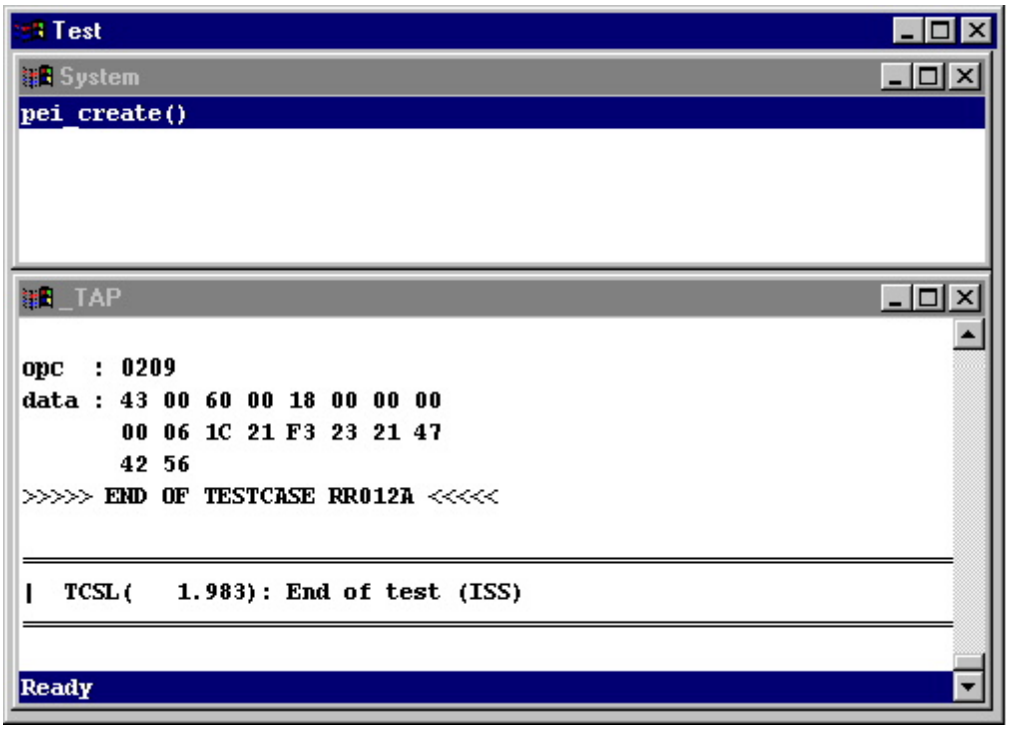
The start of the test case execution is triggered by clicking on the blue up arrow icon or by selecting the menu option <Control> <Start Test>.

If a PS is tested containing the MMI Framework, an additional window called “Dialog” is seen. This window shows the notifications of the MMI:

7 Test Case Result Evaluation

Depending on how the test case has been executed, the test case result is obtained in different ways:

If the processes TAP2 and PS have been started manually from the command line, then the test case result is displayed in the child window "_TAP" of the window "Test" (second parameter of TAP2):



The line “>>>>> END OF TESTCASE <<<<<” indicates that the test case has been executed successfully. In some environments also the two lines of text "End of test (ISS)" and "Ready" appear. Additionally, the file TEST_<entity>.PRT contains the test case results. For example, after the execution of the test case CC012A, TEST_CC.PRT may contain the following:

```
+--------------------------------------------------------------------+
| PROTOCOL OF TEST                                                   |
| Generated by TAP on Wed Mar 24 21:58:42 1999                       |
+--------------------------------------------------------------------+


Running CC012A  build Mar 24 1999 at 21:58:34 ... Passed
```

The word "Passed" indicates a successful outcome of the test case execution. Otherwise the word “Failed” is seen.

When using the Tapcaller tool for executing test cases, the test result is output in the "state" column of the graphical front end.

Appendices

A. Acronyms

DS-WCDMA	Direct Sequence/Spread Wideband Code Division Multiple Access

B. Glossary

International Mobile Telecommunication 2000 (IMT-2000/ITU-2000)	Formerly referred to as FPLMTS (Future Public Land-Mobile Telephone System), this is the ITU's specification/family of standards for 3G. This initiative provides a global infrastructure through both satellite and terrestrial systems, for fixed and mobile phone users. The family of standards is a framework comprising a mix/blend of systems providing global roaming. <URL: http://www.imt-2000.org/>