Technical Document

GSM PROTOCOL STACK

G23

TCAL – TAPCALLER

DEVELOPER DESCRIPTION

Document Number:	6-03-31-SLL-003
Version:	0.3
Status:	Draft
Approval Authority:	
Creation Date:	2002-May-07
Last changed:	2015-Mar-08 by RK
File Name:	Tapcaller_description.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
2002-May-07	JG et al.		0.1		1
2003-May-20	XINTEGRA		0.2	Draft	
2003-Aug-15	RK		0.3	Draft	2

Notes:

1. Initial version
2. New Official ID introduced

Table of Contents

List of Figures and Tables

List of 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

[TC_UDO] 06-03-31-UDO, Tapcaller Userguide

1.1 Abbreviations

TAP	Test Application Process
PS	Protocol Stack
VS	Microsoft Visual Studio
MFC	Microsoft Foundation Classes

2 Introduction

This documentation is meant for developers that are interested in the internal structure of TAPCaller. It is assumed that reader is familiar with basics of the Condat's test process and GUI programming.

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 primitve from the entity under test) and compares the received primitive with the expected one.

TAPCaller is graphical frontend for executing test cases. It should simplify the call of the TAP test tool and ease the handling of PCO. Besides, the user can easily access result files which were produced by the TAP and it is possible to call MSCVIEW.

The document is based on TAPCaller ClearCase Versions 1.1.0 or higher.

3 Overview

TAPCaller is written in C++ and uses the Microsoft's Foundation Class for external presentation. It is an single document interface (SDI) application, i.e. it allows only one open document frame window at a time (the document for TAPCaller is the test configuration consisting of paths to test tool executables, a set of test cases and so on).

Like for other SDI programmes the main classes for the application body are Document, View, Application and Mainframe classes. In TAPCaller they are named as follows:

CTapCallerApp	contains member functions for initialising of the application
CTapCallerDoc	contains data handling routines, most important part of TAPCaller
CTapCallerView	used for presentation of data defined in CTapCallerDoc, the menus and the toolbar
CMainFrame	a CFrameWnd child used for managing of the main window

The data handling (mainly in CTapCallerDoc) and representation (mainly in CTapCallerView) is not strictly split. The managing of some menus is done within CTapCallerDoc, for example.

Besides, several classes are implemented for the dialogs used in TAPCaller. These classes are:

CAboutDlg	dialog that displays information on TAPCaller (accessible with menu item <?><About TapCaller>)
CTestcaseSelectDlg	dialog used for selecting test cases and suites (accessible with <Configuration><Selection>)
CTapOptionsDlg	settings for TAP (path to executable, interface, ...), page of the "settings" sheet accessible with the menu item <Configuration><Settings>
CTestToolSelect	settings for PCO and path to PS executable, page of the "settings" sheet accessible with the menu item <Configuration><Settings>
CViewToolSelect	dialog with path to text file viewer and to MSCView executable, directory of TDS files (for loading of test titles), ..., page of the "settings" sheet accessible with the menu item <Configuration><Settings>

CProtocolInfo	dialog used for production of a protocol file, the protocol file contains the names of the test cases for the actual test session, test dates and verdicts, accessible with menu item <Configuration><Protocol>

Objects of these dialog classes are created by CTapCallerDoc when the correspondent menu item is selected. Almost all data members of CTapCallerDoc, which are imported from the dialogs, have the same or a similar name in both classes. There are CTapCallerDoc::m_nInterface and CTapOptions::m_nInterface, for example.

Three supporting classes are used in TAPCaller. These classes are:

CNotePad	most important data member of this class: list of CNotePadEntries (in member variable CNotePadList), i.e. list of currently used test cases which are displayed in the main window, used for transferring the test case list between CTestcaseSelectDlg and CTapCallerDoc
CNotePadEntry	one currently used test case with its name, the test state ("Passed", "Failed" or "Unknown"), date and time of last execution, used in CNotePad
CMyCmdInfo	class for evaluation of the command line parameters, used in CTapCallerApp

An object model of TAPCaller with the most important classes is presented on the next page in Coad-Yourdon notation.[1]

[1] The theoretic model of Coad-Yourdon is different from Microsoft's object model. Therefore the rules for representation were slightly adapted. It was refrained from a presentation of all details, e.g. CTestcaseSelectDlg is the single member of a SelectionSheet, this is not drawn to simplify matters.

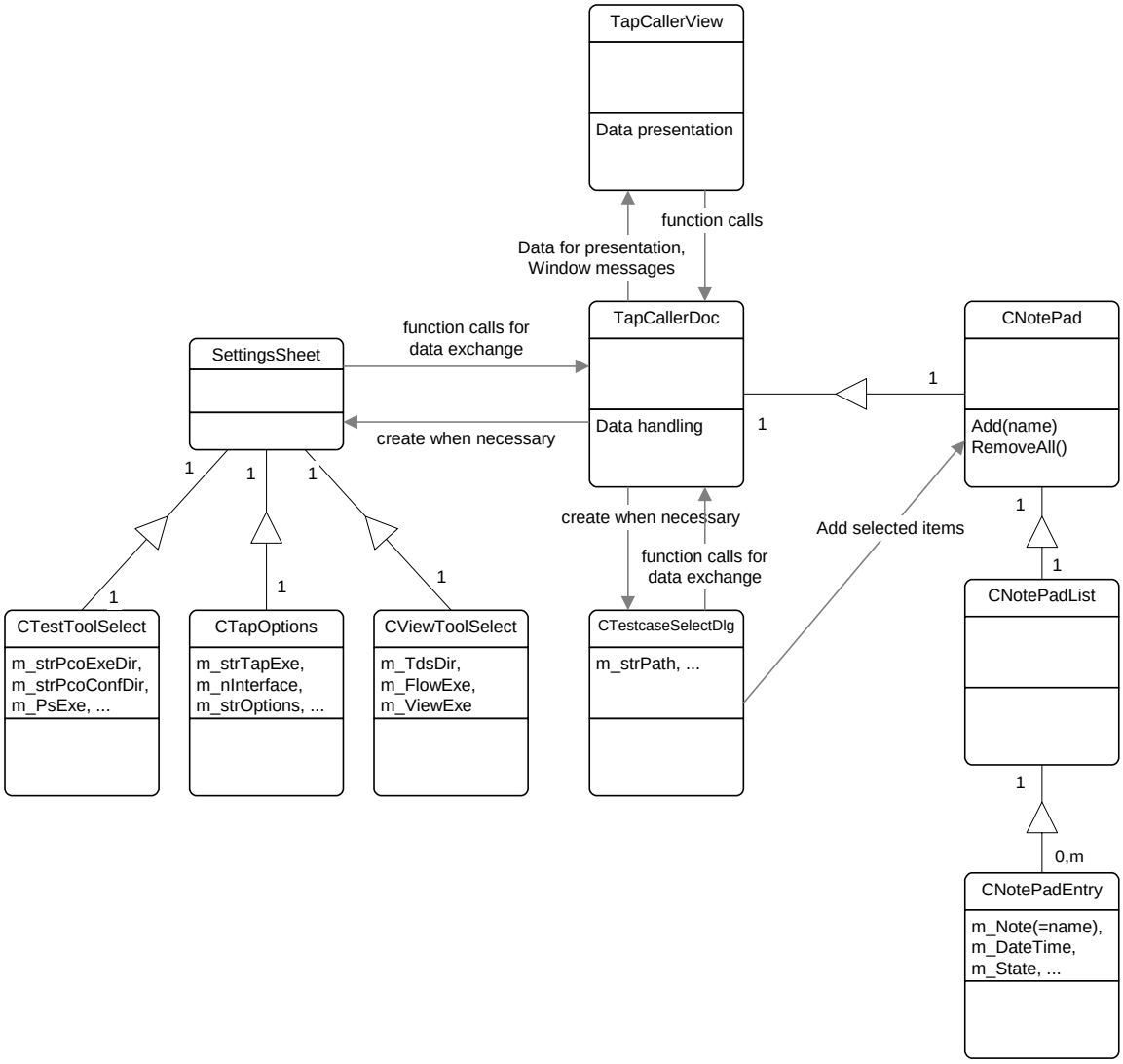

Figure 1: TAPCaller Object Model in Coad-Yourdon Notation

4 Implementation Details of TAPCaller Classes

4.1 Communication between the classes

4.1.1 Data exchange between CTapCallerDoc and Dialog Class Objects

The data exchange between the objects is mostly implemented with function calls. All dialog objects have a member variable m_pDoc, a pointer to the current CTapCallerDoc object. While creating objects of the dialog classes, CTapCallerDoc passes a pointer to itself to the created objects. Have a look at the definition of the constructor for CTestcaseSelectDlg and the creation of a CTestcase-SelectDlg object in CTapCallerDoc:

```
CTestcaseSelectDlg::CTestcaseSelectDlg(CTapCallerDoc *in_pDoc)
{ …
   m_pDoc = in_pDoc;  /*a pointer of the current CTapCallerDoc object is stored in the member variable m_pDoc*/
…
```

```
}
void CTapCallerDoc::OnConfigSelection()
{ …
CTestcaseSelectDlg  TestcaseSelectPage (this);
/*CTestcaseSelectDlg object is created with a pointer of CTapCallerDoc to itself (=this)*/
…
}
```

The member variable m_pDoc is used in the dialog objects to access public member functions (and data members) of CTapCallerDoc. With this, the dialog objects can retrieve and alter data members of CTapCallerDoc, e.g. in the OnOk/OnApply member functions of the dialog classes which are called by MFC when the 'OK' is pressed in the dialog.

4.1.2 Communication between CTapCallerDoc and CTapCallerView

Document and View Class objects have the member functions GetDocument() or GetNextView() in order to get pointers of the other object. This is used in the TAPCaller implementation, too. These pointers are used for function calls between the two classes.

Besides, programmer defined window messages are used to communicate from CTapCallerDoc to CTapCallerView while executing a test case. The window messages are sent by CTapCallerDoc when a test case is started and a test case terminates. The handling functions for these window messages change the status of test case entries (see the message maps for the mapping between command IDs and handling functions). Have a look at the message handling routines within CTapCallerView for the messages WM_SINGLE_TEST_END, WM_SINGLE_TEST_START and WM_SINGLE_TEST_UPDATE.

4.2 CTapCallerDoc

4.2.1 Process of starting a test case

TAP is started when one of the following commands are triggered: IDM_START_TEST (by menu item <Control><Start tests>, the accelerator key <Ctrl+R> or toolbar item) or IDM_TEST_START (menu item <Control><Start selected Tests> or the accelerator key <Ctrl+T>). The following chart displays the flow of actions for IDM_START_TEST, but the sequence is in principle the same for IDM_START_TEST.

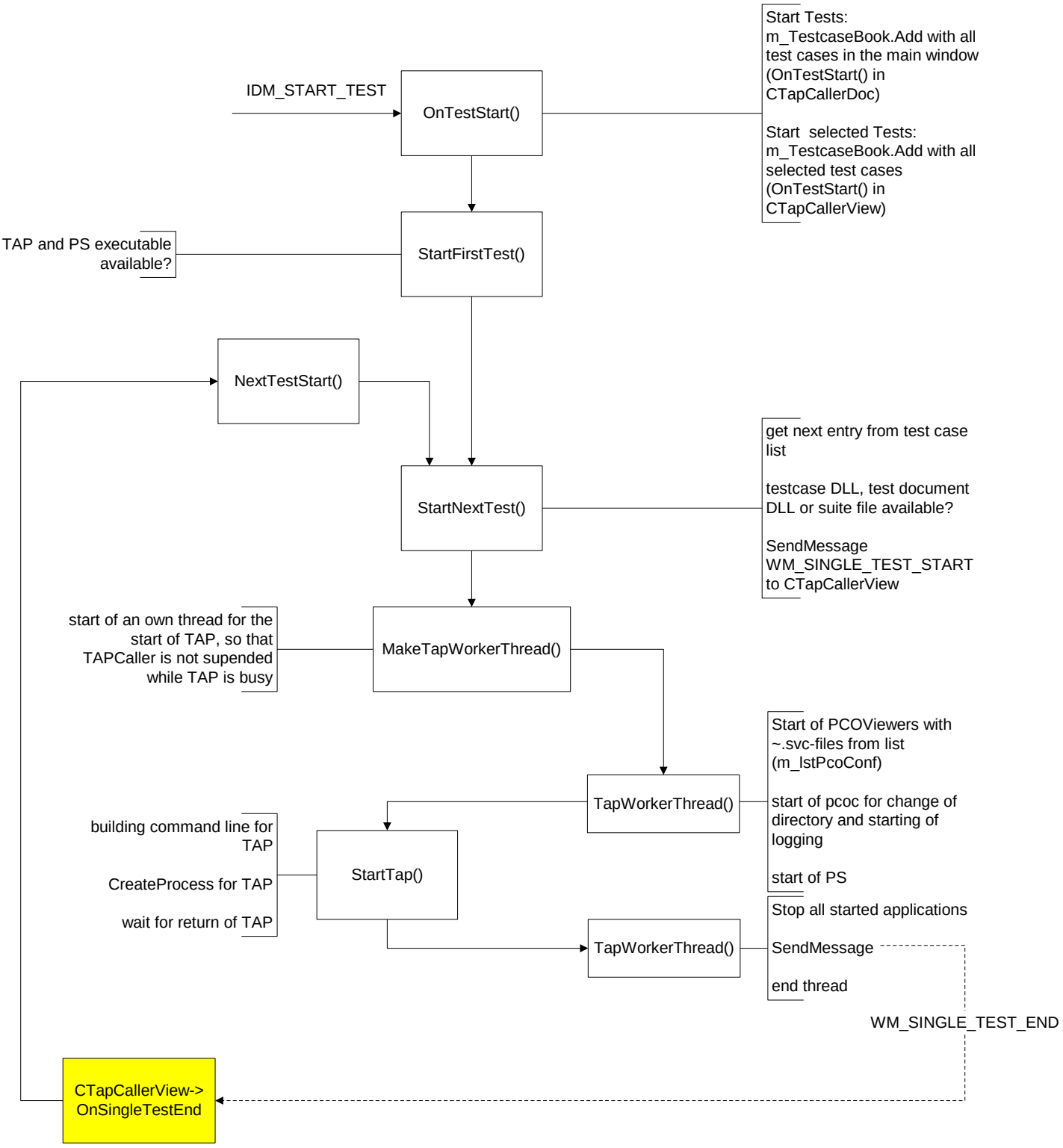

Figure 2: Sequence of functions for execution of TAP

Almost all functions are within CTapCallerDoc, except the one with yellow background. This is a member function of CTapCallerView which is started on the reception of the Window Message WM_SINGLE_TEST_END (see the message maps for the mapping between command IDs and functions to be executed). Its only task is to start NextTestStart() of CTapCallerDoc which calls StartNextTest() for the following test case. Menu and toolbar items are disabled using the m_state

member variable of CTapCallerDoc, e.g. have a look at the following code for the enabling/disabling of the 'Stop'-button of the toolbar

```
void CTapCallerDoc::OnUpdateStopTest(CCmdUI* pCmdUI)
{
   pCmdUI->Enable(m_State == STATE_TEST_ACTIVE);
}
```

The 'Stop'-button is only enabled if a test case is running (m_State==STATE_TEST_ACTIVE).

4.2.2 Serialization (storing test session data)

Serialization is used to store the test session data into a file.

Each test session data file has a format indicator as its first entry. This number is used to distinguish between the file formats. Each time a new data item is added to the test session configuration file, a new file format has to be introduced. Up to now the formats from DOCUMENT_FORMAT_VERSION1 to DOCUMENT_FORMAT_VERSION9 are defined. The newer format versions are extensions of the older ones, i.e. the sequence of storing remains the same, but there are some new variables added to the file. That is why the loading code for the test session starts with the evaluation of the format ID, and depending on this ID TAPCaller starts to read from the configuration file.

In the currently used format, after the format ID the number of used PCO configurations is saved. This is followed by the names of the PCO configuration files, PCO and TAP options, some properties of the TAPCaller window, ...

The protocol file specified in the <Configuration><Protocol> dialog is also saved while storing test session data. This protocol file contains a summary of the test session, i.e. author and title of the test session as well as an entry for each test case in the main window with its test verdict, date and time of execution and so on. It does not contain data necessary for future sessions. It is only used for documentation.

4.3 CTestToolSelect

As can be seen from Figure 1, CTestToolSelect is part of an object derived from the standard class template CPropertySheet. This object is only responsible for displaying the frame around the dialogs and buttons at the bottom of the frame. All data handling and exchange with CTapCallerDoc is done inside the dialog classes. The PCOViewer configuration is shown as example.

4.3.1 PCOViewer Configuration Selection

The configuration files for PCOViewer can be selected using two dialog elements in the middle of the dialog, one edit box for path selection and a check list box. If the edit box is changed (directly or with the path selection dialog), the check list box is updated (see the member function "void CTestToolSelect::UpdatePcoViewersList()"). For each ~.svc file found in the directory an entry is placed into the check list box. Additionally, a standard entry is the so-called "Standard Viewer". With this selection, PCOViewer is started without any parameters (in this case PCOViewer searches for the last recently used configuration file in the Windows registry or starts with its default configuration file).

When the user leaves the dialog (the OnApply function is called), the list is checked for marked items. Each of them is added to the CTapCallerDoc member m_lstPcoConf (m_pDoc-

>m_lstPcoConf.AddTail (szViewerConf)). Besides, all other necessary data members are passed via CTapCallerDoc function calls or direct change of the member variables.

4.4 CTestcaseSelect

CTestcaseSelect is the only member of a CPropertySheet object created in CTapCallerDoc when OnConfigSelection is called (with <Configuration><Selection>). In order to simplify matters the PropertySheet is not drawn in Figure 1.

If the edit box with the path to the test cases is updated (directly or with the path selection dialog), all DLLs and ~.sui files from the chosen directory are displayed. All DLLs that match test case name conventions can be selected (have a look at the member function "void CTestcaseSelectDlg::OnSelchangeTestcaseList()"). For other DLLs the "Details" button is enabled, since these DLLs may be test document DLLs that contain more than one test case. With the DetailsofDLL member function explained in chapter "4.4.1 Adding test cases from a Test Document DLL to the test case list" these test case are added to the check list box.

When a suite file is selected, pressing the "Details" button results in calling the member function DetailsofSuite described in chapter "4.4.2 Adding suites to the test case list". All available suites from this file are added to the list.

The list of test cases is not passed directly from the CTestcaseSelect object to the CTapCallerDoc object, but an object of the CNotePad class is used. When user leaves the test case selection dialog, the check list box is checked for marked entries. Each of them is added to test case list in the CNotePad object (m_TestcaseBook.Add (szTestcase, bUsed)). The CNotePad object is used in CTapCallerDoc for displaying all selected test cases in the main window.

4.4.1 Adding test cases from a Test Document DLL to the test case list

When the user presses the "Details" button for a DLL, the function DetailsofDLL is called. This results in a call of AddFunctions, a member of CTestcaseSelect that opens the specified DLL as text file, reads the necessary data from the DLL file header and tries to locate the export table of the DLL. From this export table all exported functions of the chosen DLL are tested whether they match the name pattern for test cases. If so, they are added to the test case list. This part of TAPCaller is well commented in the source code of TAPCaller, since the structure of a DLL is not easy to understand.

AddFunctions() is implemented with standard C routines due to the fact that the function was developed as command line tool and afterwards integrated into TAPCaller.

For further information on the structure of DLLs have a look at "windows.h" or more detailed descriptions of the common object file format (COFF).

4.4.2 Adding suites to the test case list

Like test document DLLs, suite files are opened as text file. This file is searched for lines beginning with "SUI", since all suite names to be used externally begin with this character string. The names are added to the to the check list box and are handled like normal test cases, except in the calling of TAP (the command line is adapted in this case).

5 Other hints

Build possible with 2 methods (VisualStudio or makefile)

Non-compatibility of VS 5.0 and 6.0 debug libraries

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/>