Technical Document

GSM PROTOCOL STACK

G23

PCO2 – TRACING ENVIRONMENT

USER GUIDE

Document Number:	06-03-35-UDO-006
Version:	0.6
Status:	Draft
Approval Authority:	
Creation Date:	2000-Dec-15
Last changed:	2004-Sep-01 by Ronny Kiessling
File Name:	pco_userguide.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.

Table of Contents

0 Document Control

0.1 Change History

Date	Changed by	Approved by	Version	Status	Notes
2000-Dec-15	RK		0.1		1
2001-May-15	RK		0.2		2
2002-Sep-30	RK		0.3		3
2003-May-21	SPRK		0.4	Draft	
2003-Aug-18	RK		0.5	Draft	4
2003-Sep-17	RK		0.6	Draft	3

Notes:

1. Initial version
2. Standard Viewer doc added
3. updated
4. New official Document ID introduced

0.2 List of Figures and Tables

0.3 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
[PANEL]	8415.014.00.105, February 7, 2000, Panel – PC Test Application
[XPAN]	06-03-36-UDO, xPanel Developer Description (xpan_userguide.doc)
[MOAN]	06-03-53-UDO, MoanBtn – Instant GUI-problem Informer (mbtn_userguide.doc)
[XM]	06-03-55-UDO, XM –GUI-frontend for GPF m.bat (xm_userguide.doc)
[PCO2_D]	06-03-35-SLL, PCO2 – Developers Description (pco_description.doc)
[FRAME]	8434.100.02.001, January, 2002, Frame Users Guide (frame_users_guide.doc)
[PCO_INTRO]	Testing with xPanel & PCO2 (pco_intro.ppt)
[PCO_FILTERS]	Filter levels with PCO2 (pco_filtering.ppt)
[PCO_TCGEN]	Tracing for TCGen (pco_tracing4tcgen.ppt)
[STR2IND]	February, 2002, Compressed/Binary tracing (str2ind_userguide.doc)
[TCGEN]	06-03-32-UDO, Test Case Generator (tcgen_userguide.doc)

0.4 Abbreviations

ACI	Application Control Interface (AT Commands)
CC	Call Control
DL	Data Link Layer
FAD	Fax And Data
G23	The Condat implementation of Layers 2 and 3 of the GSM Protocol Stack
G23 Target System	Hardware which executes G23
GMM	GPRS Mobility Management
GRR	GPRS Radio Resource
L1	Layer 1
L2R	Layer 2 Relay
LCD	Liquid Crystal Display
LLC	Logical Link Control
MM	Mobility Management
MMI	Man Machine Interface
MOC	Mobile Originated Call
MTC	Mobile Terminated Call
PC	Personal Computer
PCO	Point of Control and Observation, file format for storing whole test sessions
PL	Physical Layer
PPP	Point to Point Protocol
RLP	Radio Link Protocol
RR	Radio Resource
SIM	Subscriber Identification Module
SM	Session Management
SMS	Short Message System
SNDCP	SubNetwork Dependent Convergence Protocol
SS	Supplementary Services
SVE	Standard Viewer Entries, files format for storing all traces/hexdumps currently shown in a specific Standard Viewer
T30	T30 (FAX Protocol)
TCGEN	Test Case Generator
PIN	Personal Identification Number
RS232	Serial Communication Standard
Target System	Shortened form of 'G23 Target System'
UART	Universal Asynchronous Receiver Transmitter

For further abbreviations see e.g. the documents under ClearCase control in \g23m\condat\ms\sap.

0.5 Terms

Entity	Program which executes the functions of a layer
Message	A message is a data unit which is transferred between the entities of the same layer (peer-to-peer) of the mobile and infrastructure side. Message is used as a synonym to protocol data unit (PDU). A message may contain several information elements.
Primitive	A primitive is a data unit which is transferred between layers on one component (mobile station or infrastructure). The primitive has an operation code which identifies the primitive and its parameters.
Service Access Point	A Service Access Point is a data interface between two layers on one component (mobile station or infrastructure).

1 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/AT Command Interface,
- MMI and MMI Framework (MFW) and
- Test and integration support tools.

This document is the User Guide for the tool “PCO”. This guide is based on the assumption that the user is familiar with the basic operation principles of mobile phones. It is meant to give an introduction of the functionality and the co-ordination of the sub-tools making up the new PCO, and to explain several important issues in detail.

PCO is a tool that finally provides an interface for tracking all kind of traces and primitives.
Originally it was designed as a TI-internal tool only but there are already some customers using the “old” PCO as well. The TAP and the old PANEL also contain parts of it.

For the new PCO the concept has been completely updated: Unlike former PCO-solutions the current version does not (directly) depend on the FRAME. Furthermore there are now at minimum three stand-alone executables - a server, a controller and one (or more) viewer(s).

Currently there are only Win32-versions (GUI supported or command line based) available.

PCO does also include moan button support (see [MOAN]), if you got the moanbtn.dll with the package (internal TI workers with ClearCase automatically get it).

PCO is typically used in combination with the xPanel (see [XPAN]) or the new TAP2.

2 General Application Manual

2.1 Environment / Installation

To use pco some environmental constraints have to be taken into account.
For use under Windows:
You’ll have to make sure that several DLL/EXE-files are available to the system. In the TI development directory structure you can find them in „<View>/GPF/BIN” :

- CMS.dll, ccddata_load.dll, frame.dll, misc.dll, tif.dll, tst.exe
- Ccddata_dll.dll (if you want to use CCD-support)

So just make sure „<View>/GPF/BIN” is in your PATH-variable.
The “Ccddata_dll.dll” will be rebuild every time you call “makcdg”.

To avoid periodically reconfiguration you should further provide an ini-file or modify the default one: „<View>/GPF/cfg/pco.ini” (see 2.3 for details). It will be modified automatically after closing PCO.

2.2 Getting started

For your convenience a dedicated batch file comes with the PCO which starts a minimum set of components to work meaningful. (it actually just calls the GUI-Controller to load the configured test environment – see 3.2.6)

To start PCO the first time just call pco2.bat. Now you should see the PCO-server in the corner right down (or minimized to system tray), the PCO-controller in the corner right up and 2 standard viewers somewhere on screen (its position can be stored). Additionally the xPanel will be started (see [XPAN]). Now start your stack and you should see traces in the standard viewer. (If not, you probably have to change the communication settings – see 3.2.4)

See [PCO_INTRO] (pco_intro.pps) for a short introduction with screen shots.

2.3 Command line parameters and ini-files

From the command line you can specify some environmental options for the pco-components.
The command line parameters of the convenience batch file pco2.bat are as follows:

- h	... display all parameters
- nogui <sessionname>	... start server without GUI to record a session (+ GUI-viewer)
- nogui replay <sessionname>	... start server without GUI to replay a session (+ GUI-viewer)
- tst_com{1|2}	... configure communication via test interface on com port 1/2
- at_com{1|2}	... configure communication via at interface on com port 1/2
- sim	... configure communication via shared memory
- ti_mux	... configure communication via TI multiplexer
- {ccd|no_ccd}	... load specific default test environment with/without requiring ccddata_dll.dll

See 3.1.1, 3.2.2 and 3.3.1.1 for component specific information.

If you don't give any parameters the default ini-file "pco.ini" will be parsed if one exists in the current directory or in the "..\cfg"-directory of the executable.
In an ini-file only lines of the following format will be interpreted:
'['<component>']'
<parameter><whitespace>'='<whitespace><Value>

There exist some general settings applicable to all PCO components:
[General]

- CCDDataPath … path to the ccddata-dll which shall be used for interpretation/decoding (e.g. ccddata_dll.dll)
- DataSize ... size of an data element in queues for traces/primitives in bytes (e.g. 1400)
- ServerName ... internal name of the used PCO server (PCOS, L3SRV, …)

An overview of the available component specific parameters is given in the sections 3.2.3, 3.2.3 and 3.3.1.2.
You can find examples in the default pco.ini file.

2.4 File formats supported by PCO

This chapter contains information about the different file formats which can be used together with PCO.

2.4.1 *.pco – PCO-logfiles (test sessions)

Description:
PCO-logfiles contain all data received via test interface during a test session – independent of any filter settings in connected viewers (see also [PCO_FILTERS]). If no str2ind-table was found at test time it contains the pure indices, so during later replay a correct str2ind-table can still be applied. If a matching table was found, the logfile contains the decompressed traces.
Moreover, the ccddata-DLL used during testing is attached to the file for later usage.

Created by:
… PCO-Server (see 3.1) which has to be controlled to start/stop logging either by the command line PCO-Controller (see 3.2.2) or by the GUI-Controller (see 3.2.1 and [PCO_INTRO] chapter "Logging & Replay"). Per default the logfiles will be stored into folder ..\bin\testsessions.

Usage possibilities:
PCO logfiles can be used in various ways, e.g., for replay through all connected viewers via the PCO-Controller (see 3.2.1 and [PCO_INTRO] chapter “Logging & Replay”), for filtered examination by loading into a Standard Viewer (see 3.3.1) or even to generate new TDC testcases with TCGen (see [PCO_TCGEN]).
Furthermore, once loaded into a Standard Viewer the filtered content can be exported into other formats like .sve, .dbg or .txt(see 3.3.1).
Parts of a logfile can be stored into new ones via the GUI-Controller (see 3.2.1) or the Standard Viewer (see 3.3.1).

2.4.2 *.sve – Standard Viewer Entries

Description:
As the name implies .sve files contain entries (traces or hexdumps of duplicated primitives) of a PCO Standard Viewer which could be seen there at the time of saving. That means, e.g., all traces from entities which were not in the watch list of the particular viewer can not be reproduced via such a file. It also does not contain any information about the ccddata-DLL used and if no str2ind-table was found at test time the .sve-file will contain the STR2IND error messages as displayed in the viewer.
On the other hand, evtl. added entry comments or dedicated colors for some entries (not the general sender colors) will be saved, too.

Created by:
… PCO Standard Viewer (see 3.3.1). If no or only one entry is selected all are stored, otherwise only the selected ones will be taken into account.

Usage possibilities:
This format has been introduced in the early days of PCO on request by many testers – to have an easy way to select certain (commented) traces, store and send them to other people. .sve-files can only be loaded into a Standard Viewer, but from there export into other formats is possible, of course. Anyway, it is highly recommended to log into .pco-files as well to save all emitted data and forward the decision of which filter shall be applied to replay time.

2.4.3 *.svc – Standard Viewer Configurations

Description:
.svc-files contain all settings you can make inside the PCO-Standard-Viewer application including, e.g., size and position of the window, selected filters and colors or str2ind/ccddata-config sets (see 3.3.1.6 and following for more infos). They can also contain entries of the Viewer (see .sve-files above) if the option “Store entries with configuration” in the <File> menu has been enabled.

Created by:
… PCO Standard Viewer. Per default the viewer will automatically store all changes in the currently configured svc-file without questioning unless the options “auto save configuration on exit” has been disabled.

Usage possibilities:
.svc-files are used to store and resample user settings in the various Standard-Viewer instances used. Per default to configurations (main.svc and syst.svc) are used by the two viewers contained in the initial test environment (see 3.2.6). You may specify a certain .svc-file as parameter to the Standard-Viewer on the command line (see also 3.3.1.1) or, e.g., drag and drop such a file to a running viewer to import the stored settings.

2.4.4 *.dbg – DeBuG traces

Description:
Files with the extension .dbg are ASCII-files containing traces formatted in a historical style.
They are still supported for backwards compatibility reasons but their usage is not recommended anymore.

Created by:

… PCO Standard Viewer or PCO-Server. The viewer supports menu entries under <File><Export> to create one .dbg file containing all selected traces (or all if only one or zero are selected). A second menu entry can be used to export the traces into separate .dbg files for each sender entity.
The server will optionally create separate entity-.dbg-files together with a .pco-logfile in the same directory. To request this you may directly change the ini-file (see 3.1.2) or modify the settings in the GUI-dialog.

Usage possibilities:
.dbg-files can be used for direct examination in any text editor or they can be re-imported into a PCO Standard-Viewer (e.g., by drag and drop).
Furthermore the tool MFCVIEW can interpret them and generate a graphical message flow diagram.

2.4.5 *.txt – ASCII output

Description:
.txt files contain the content of Standard-Viewer entries in ASCII format. The various parameters (like time, sender etc.) are separated by tabulators. In case of duplicated primitives the hexdump is contained but the interpreted structure elements can also be exported, optionally.

Created by:
… PCO Standard Viewer. You find the corresponding menu entry under <File><Export>.
In the dialog under <View><Options> it can be decided which column (like time or sender) shall be exported and up to which level duplicated primitives shall be expanded.

Usage possibilities:
.txt-files can be used for direct examination in any text editor or for any kind of post-processing.

2.4.6 *.tab – Str2Ind tables

Description:
Str2Ind tables are ASCII formatted and contain a list of indices and corresponding trace texts.
Such tables must never be edited manually!

Created by:
… PS build process. When using, e.g., BuSyB you'll find them in ..__out__\<variant>\trace.

Usage possibilities:
Str2ind tables are needed to decompress trace indices coming out of a protocol stack under test.

3 Application Manual for the specific components

3.1 The server

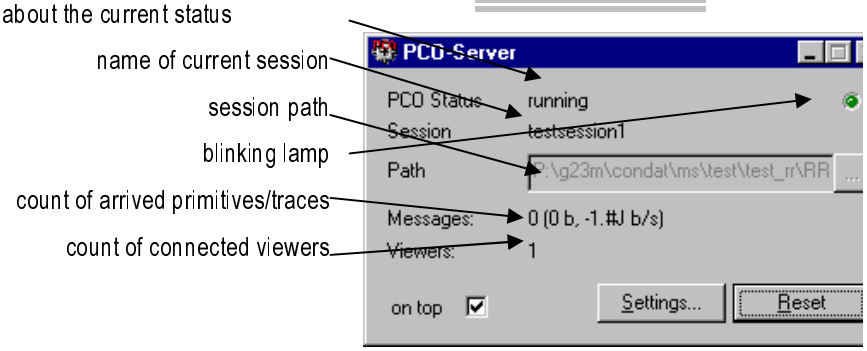
The PCO server is the fundamental part of PCO2. It receives traces and redirected primitives from a testinterface-entity running on top of the FRAME. This is provided as a stand alone executable: tst.exe.

There are currently two implementations – a command line server (pcod.exe) and a GUI-server for win32 (pco_srv.exe, pco_minisrv.exe without CCD support).
They can not be started in parallel !

The command line executable will show *"PCO server waiting ..."* when started and print out strings like *"... connected to .."* if a viewer connects or *"... testsession ... started"*. Whenever a primitive or trace arrives a '.' will be echoed.

The GUI-server can appear in two ways:

- minimized to the system tray where it will blink on arriving primitives/traces
- as a dialog window with more information about the current status

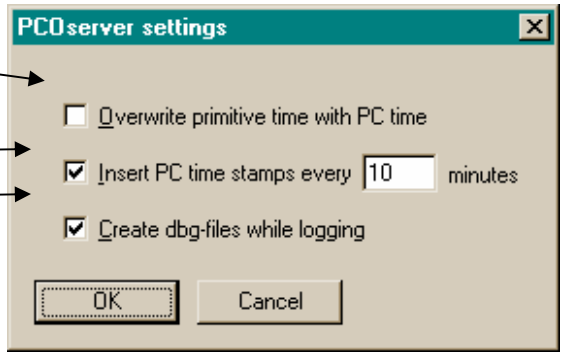


The colors of either the system tray icon or the lamp in the dialog depend on the server status:

- red: server is stopped (that is he doesn't log any primitives but still forwards them to connected viewers)
- green: server runs in logging mode and stores traces/primitives in the selected session-file (.pco-file)
- yellow: server runs in logfile mode and awaits requests from viewers to distribute the loaded session

By selecting the "Settings..."-Button further settings can be specified:

- overwriting of the time inside primitives with the PC time while receiving it
- inserting of periodically time stamps as traces
- create historical dbg-files together with the .pco-file



The "Reset"-Button will reset the internal state of the PCO-Server.

3.1.1 Command line parameters

The command line server understands several parameters which can be listed by calling "pcod –h":

- -i <ini-file> ... use the specified ini-file
- -ver ... print version

3.1.2 Ini-files settings

In the pco.ini file (see 2.3) you can specify several server settings. Most of them can be changed using the GUI-server as well.

[Server]

- Tray {0,1} .. specifies, if the GUI-Server should be minimized to system tray
- TopMost {0,1} .. specifies, if the GUI-Server should be on top
- TestSessionPath .. path where the testsessions (.pco-files) will be stored by the server (shouldn't be on a ClearCase-View because of performance reasons)
- DataQueueSize ... count of elements in data queue between testinterface and PCO server
- SetTime {0,1} .. specifies, whether the server should set new time values in the received primitives (new time values will reflect the current PC time in milliseconds since 1970)
- TimeStampPeriod ... period in minutes the server shall insert time stamps as traces
- DontCreateDbgFiles {0,1} .. specifies whether historical dbg-files shall be created together with the .pco-file during logging

3.2 The controller

The main purpose of the PCO controller is to change the status of a running PCO server and to send SYSTEM-primitives to the stack via the server. The GUI-version furthermore hosts the test environment and configures the type of communication with the protocol stack.
The command line controller (pcoc.exe) can nicely be used in batch files. See 3.2.2 for all suitable parameters.

Command line and GUI version may be used in parallel.

3.2.1 GUI interface

The GUI-controller (pco_ctrl.exe) is currently only available for win32. It will show up as a dialog window in the right upper corner of the screen.

Change test session directory and request session names from server

To start a test session just enter a session name (or select an old one from the pull down menu) and push the “Start logging”-Button.
The server should change into green (running) mode.

To replay an existing logged session switch to replay mode (via the uppermost button), select a session name and push “Replay”.

The server should change into yellow (logfile) mode and connected Viewers will show the logged traces/primitives.

Pressing the “TCGen...”-Button will provide you with an GUI-Interface for using the test case generator with the selected session (see [TCGEN]).

Via these buttons you can activate or configure the whole test environment. “Activate” will bring all windows of the test environment into foreground. “Add viewer” starts a new PCO-Viewer window which may be added to the environment.
“Configure... ” opens a dialog for adding/removing more applications to the start-list-file (see 3.2.6).

Finally via the “Restart”-button you can restart the whole test environment, that means all applications listed in the start-list-file (see 3.2.6).

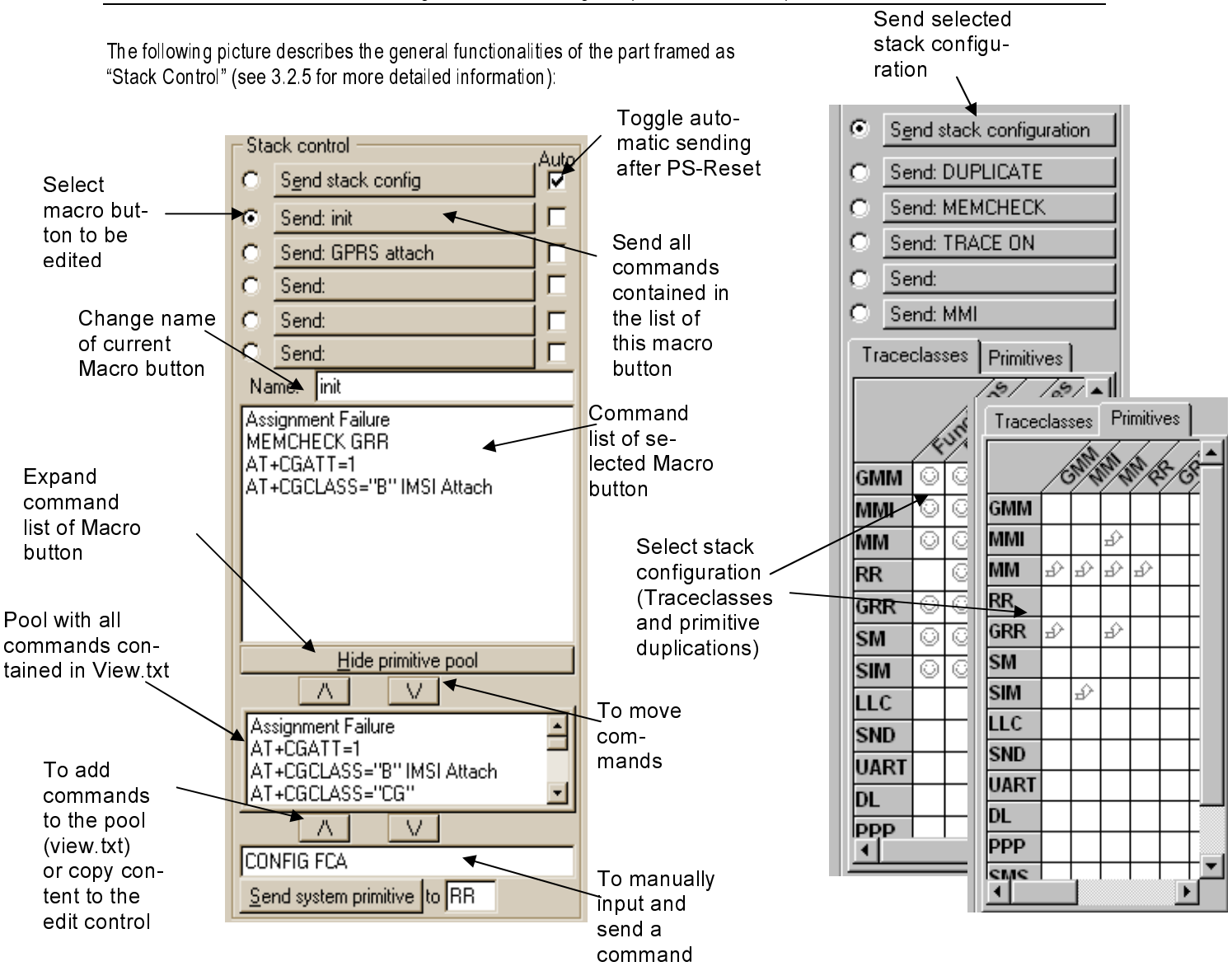
At the very bottom of the controller you’ll find these buttons: 

Use “Exit” to shut down the PCO-Controller and the test environment.

Use “Communication ...” to open a dialog for setting up the communication type to be used to connect to the PS. (See 3.2.4 for details).

Via “?” you’ll get version information.

The following picture describes the general functionalities of the part framed as "Stack Control" (see 3.2.5 for more detailed information):

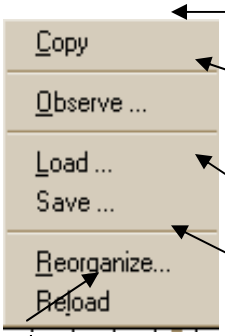


By clicking with the right mouse button on the macro-lists or the matrix one of the following context menu will appear:

Copy the content of the current macro-list into clipboard

Rename currently selected pool entry

Delete currently selected pool entry



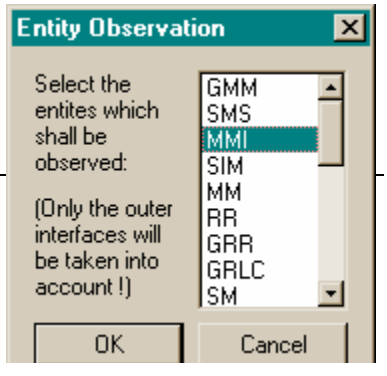
Copy content of the matrix to clipboard

Open observation dialog

Load/save matrix from/to .xml-file

Open reorganize dialog

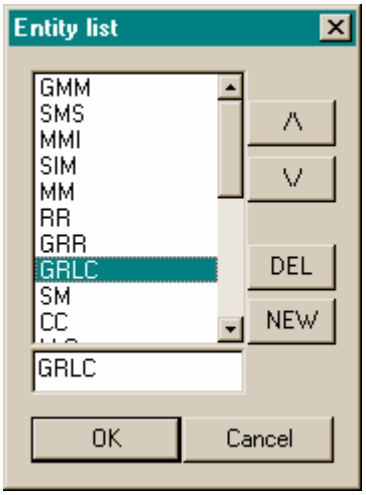
Reload ccddata information to evtl. disable matrix entries



The observation dialog is available only if a ccddata-library in version 1.6.1 (or higher) is used (see 3.3.1.8).

It provides support for selecting all necessary primitive duplications (see also 3.2.5) necessary to observe the "outer" SAPs of dedicated entities. After closing this dialog via ok the calculated settings will replace the current ones in the matrix.

This mechanism is useful if you e.g. want to create logfiles for later usage with the testcase generator (see [TCGEN] and [PCO_TCGEN]).



The reorganization dialog is always available.

It can be used to add/remove entities to/from the matrix or to change their order.

Changes in this dialog have the same effect as directly editing the matrix-file (default: pco_stack.xml, see also 3.2.3

3.2.2 Command line parameters

The command line controller understands several parameters which can be listed by calling "pcoc –h":

- spath <path> ... set <path> as the server session path to store .pco-files
- start <testname> ... start a test session named –testname-
- stop ... stop a currently running test session
- open <logfile name> ... open a logfile (stored test session)
- distrib ... make the server forward a loaded session to all connected viewers
- send <receiver> <text> ... send a SYSTEM-primitive -text- to –receiver- (Entity name)
- msend <ASCII-File> ... send multiple SYSTEM-primitives specified in an ASCII-file with entries like:
  #comment
  <receiver> <command-text>
- close ... close currently opened logfile
- exit ... exit a currently running server
- -n <server-queue-name> ... use a different server queue name
- -ver ... print version
- info [<logfile name>] ... print info about current server state/ specified logfile

3.2.3 Ini-files settings

[Controller]

- NoScreenBoundary .. {0|1} if set, the controller can be moved outside the screen boundaries
- ConfigsFile .. (see 3.2.6)

3.2.4 Test interface configuration (Communication set up)

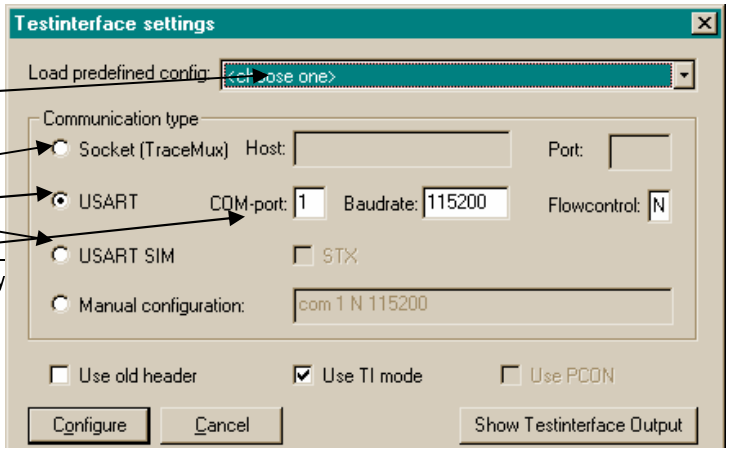
To set up the communication type to be used (the type of connection with the PS) the following dialog (started by the middle button at bottom) may be used:

Inside it you can do the following:

- Select a predefined configuration

or

- Select the general type and



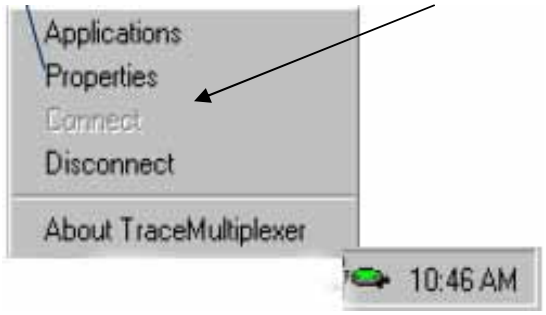
specify individual parameters
like hostname or com port

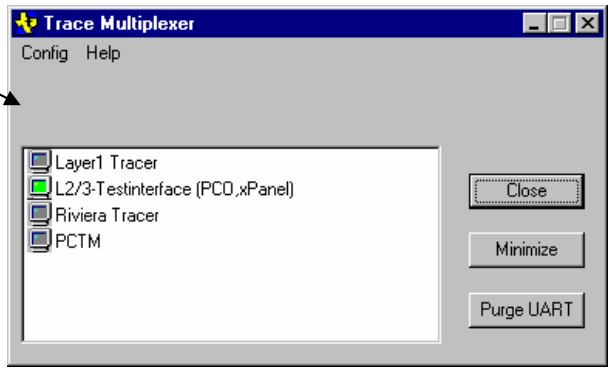
- or manually specify parameters for the tst.exe
- and additionally set some parameters

After pressing the “Configure”-button all needed tools (like TraceMultiplexer) will be started and the test interface will be (re-)configured.

For debugging purposes the output of the test interface executable can be shown/hidden.

When using the TraceMultiplexer you can see whether the connection to it succeeded by the green monitor symbol. (old version 1.0) or via the applications menu (new version 1.1).



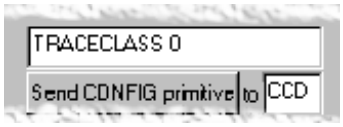


The TraceMultiplexer itself has to be configured with the COM port you are using, a baudrate of 115200 and no flow control.

3.2.5 SYSTEM-primitives, AT-commands and the primitive-file format (view.txt)

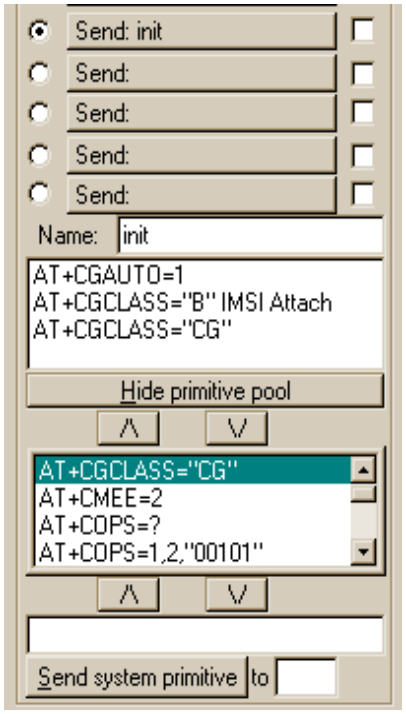
SYSTEM-primitives are ASCII-strings sent to an entity and either interpreted by the FRAME or the entity itself. By sending such a string you may e.g. set the trace class of an entity, duplicate primitives to route them outside the PS to the test system or sent an appropriate CONFIG-message to an entity, which will be interpreted in its pei_config()-function. By sending for example "CONFIG AT+CFUN=1" to MMI you can forward AT-commands to the protocol stack. See [FRAME] for detailed information about SYSTEM-primitives.

In the GUI-PCO-Controller you have a lot of possibilities to send such ASCII-strings.



It is possible to type in SYSTEM-primitive text directly in the Edit-box at the bottom and send it to a specified entity.

As described before you can specify a file with SYSTEM-primitives in the ini-file of the controller. This primitives will be shown in the pool list box from where you can copy the ones you need into on of the 5 macro-boxes (you only see one box at a time, selected by the radio-choice above).
By pushing a macro button (which can be given any name you like) all selected SYSTEM-primitives will be sent to the server and from there to the protocol stack.
By double clicking a dedicated primitive in the Macro list box you can send this one separately.

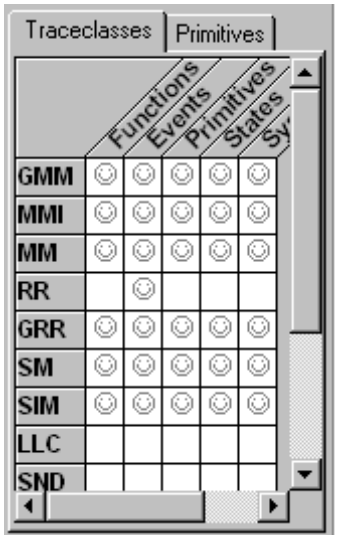


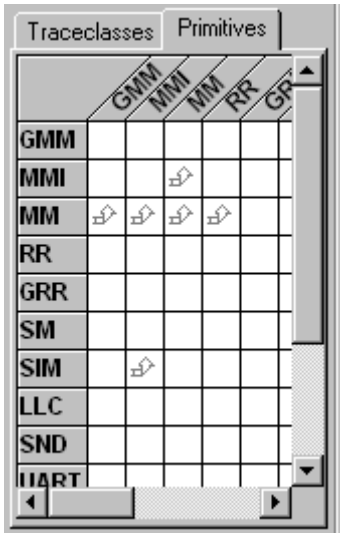
The syntax of an entry in the primitive-file (which can be named anyhow, but "view.txt" is well known from the old PANEL) is as follows:
- any empty and any line starting with a '*' is a comment
- an primitive entry goes like:
  <description>
  <config text>
  <name of entity to send to>

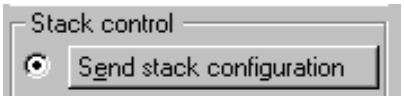
\GPF\cfg\view.txt.default is a demo version of such a file.

IMPORTANT! When editing a view.txt-file you should only APPEND new entries since the controller only remembers the position of chosen SYSTEM-primitives. Inserting entries will result in a confusion of your macro button config!
Editing is possible via the GUI (context-menu and /\\/-buttons) as well.

A third possibility is to select trace classes or primitive duplications by using the stack configuration matrixes and the corresponding button:







Every configured detail is stored in a stack configuration file which is XML-formatted. The default file is pco_stack.xml but can be changed in the ini-file (see 3.2.3).

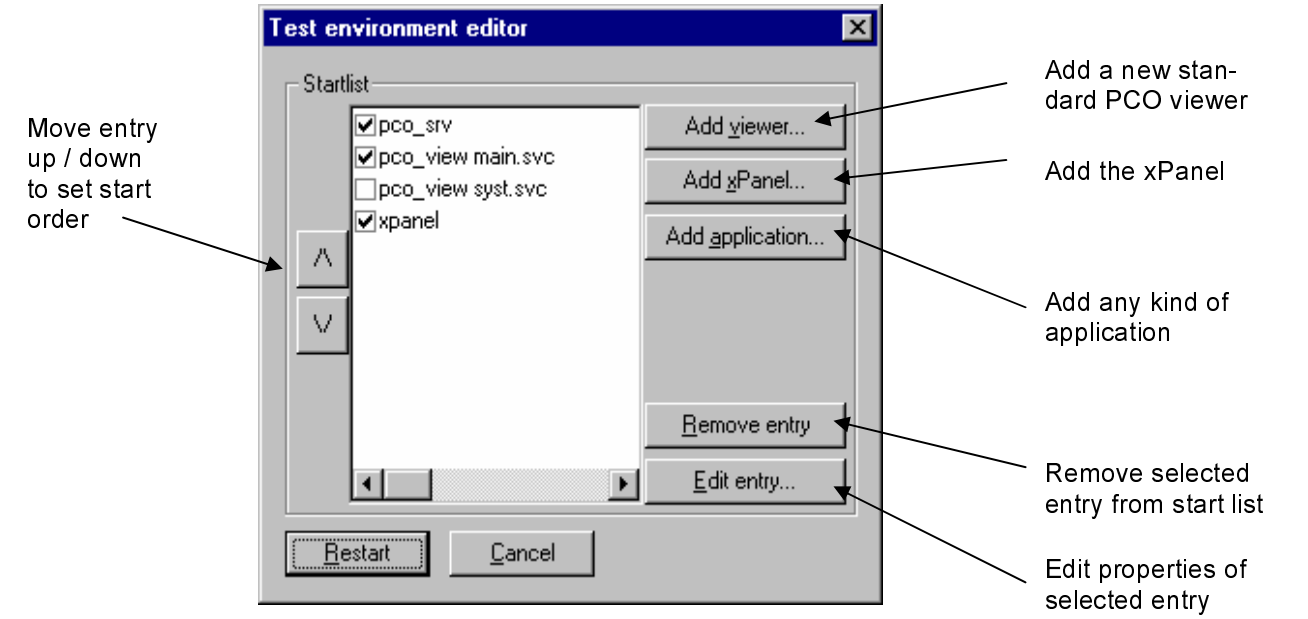
3.2.6 Test environment

- PrimFile .. file with predefined SYSTEM-primitives for Controller (usual "view.txt", see 3.2.4)
- Primlist .. set automatically
- MacroId .. set automatically
- MacroName .. set automatically
- Autosend .. set automatically
- Primtext .. set automatically
- Primreceiver .. set automatically
- StartList .. path to the start-list-file (see 3.2.6)
- StackConfigFile .. path to xml-file containing the stack configuration (traceclasses etc.)
- TSTHeader .. {new|old} – test interface header type
- TSTTiMode .. {0|1} if set, test interface will be configured in TI-Mode
- TSTStx .. {0|1} if set, test interface will be configured in STX-Mode
- TSTCommunication .. communication type (sim, socket, com {1|2|..},file <fname>... – see 3.2.4)
- NoAutolog .. {0|1} if set, controller will not start autologging upon launch
- NoRenameAfterLogging .. {0|1} if set the rename option after stop of logging is not offered
- NoCcddataFromLogfile .. {0|1} if set a ccddata-DLL contained will not be used during replay

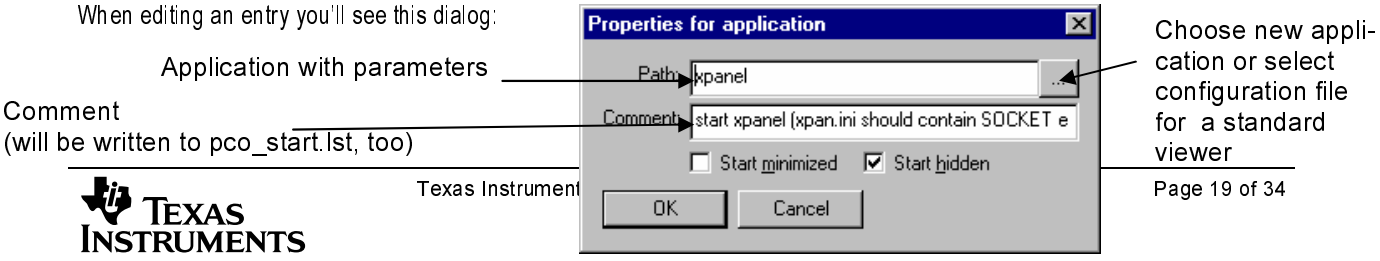
The GUI version of the PCO controller supports starting a whole test environment. That means a list of applications specified in an ASCII-file will be started when starting the controller. The name of this start-list-file is specified in the pco ini-file (see 3.2.3). Every line in the file contains the path to an application. You can use the following prefixes: -

- min ... to make an application starting minimised
- hide ... to make an application starting hidden
- # ... to deactive an entry

You may edit this start list from the GUI by selecting "Configure .." from the "Testenvironment" section:



When editing an entry you'll see this dialog:



Choose start behaviour of application 

3.3 Viewers

The concept of PCO is to be open for any kind of viewer front-end for the traces/primitives which applies to minimal requirements. See [PCO2_D] for detailed information on how to write your own PCO viewer.

The following paragraphs will give an introduction to already existing viewers.

3.3.1 The standard viewer

The so called "Standard viewer" is just the first viewer which was available. It is a MFC-based WIN32-Viewer, available in two versions: with CCDEdit-support (pco_view.exe) and without (pco_miniview.exe). The first can use a ccddata-DLL to interprete duplicated primitives/air messages. Such a DLL has to be build with the matching cdg-files (if not delivered as well).

If you use the pco2.bat this viewer will be started automatically.

3.3.1.1 Command line parameters

The following parameters may be passed to the command line of pco_view.exe:

- -h … display information about the command line parameters
- -new … create new svc-file (StandardViewerConfiguration file)
- <.svc-file> … load specified svc-file
- -i <ini-file> … uses specified PCO ini-file
- -open <pco-logfile> … request the specified PCO logfile (*.pco) from PCO-server
- -import <sve-file> … import entries from specified sve-file (StandardViewerEntry file)
- -impcfg <xml-config-file> … initially import all settings and filter configuration from specified XML-file
- -mobname <mobileId> … use specified mobile ID (supported if used with LTS-Server only)

3.3.1.2 Ini-file settings

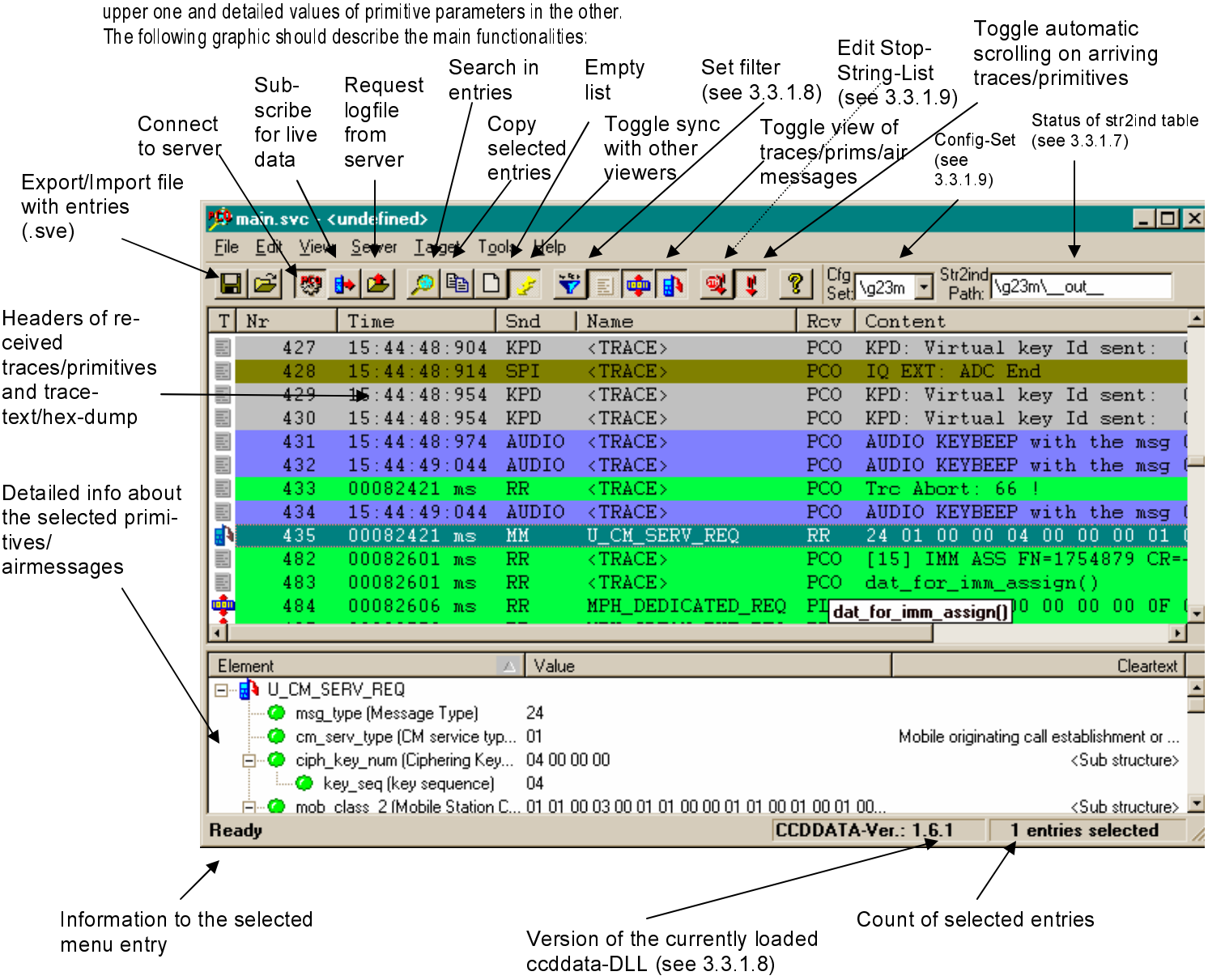
Beside the general settings (see 2.3) some specific ones can be selected in the PCO ini-file:

[Viewers]

- Str2IndPath .. path which will be recursively searched for a matching str2ind-file (*.tab) (see 3.3.1.7)
- DataQueueSize … number of elements in queues for traces/primitives

3.3.1.3 Main Windows and Toolbar

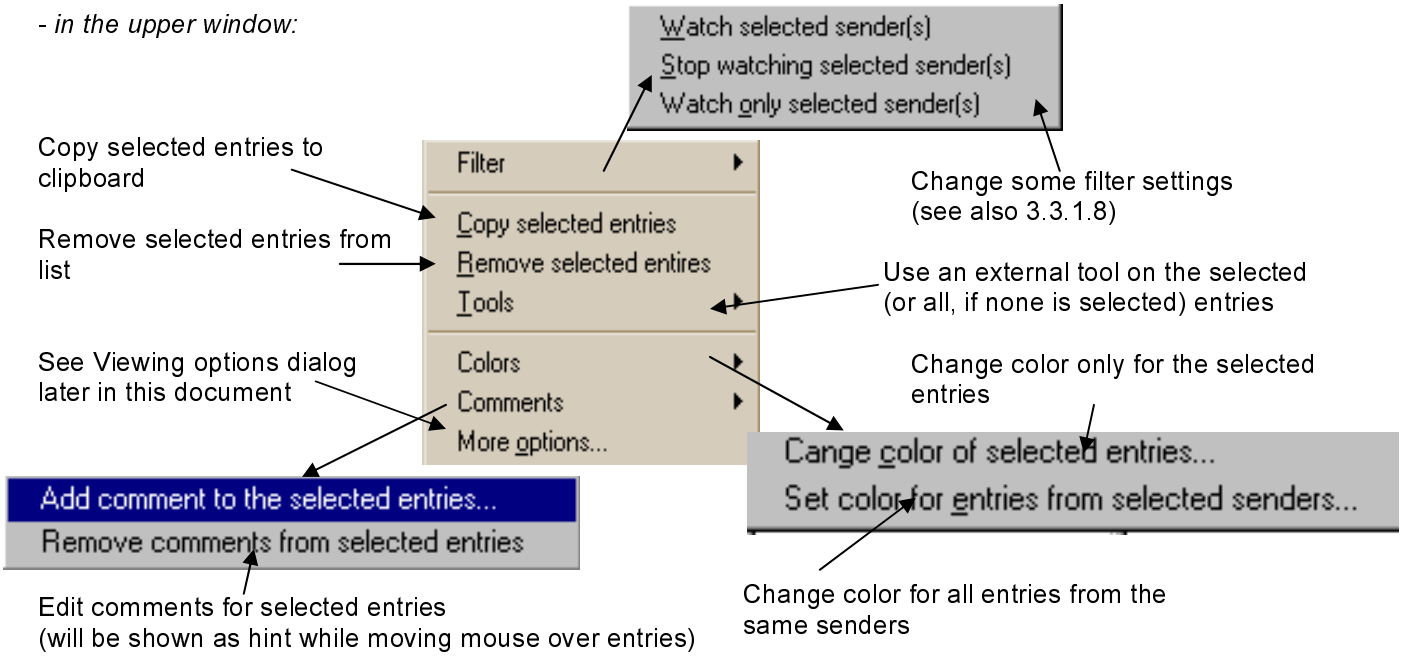
The display of the standard viewer is splitted into 2 main windows with general information about traces/primitives in the upper one and detailed values of primitive parameters in the other.
The following graphic should describe the main functionalities:



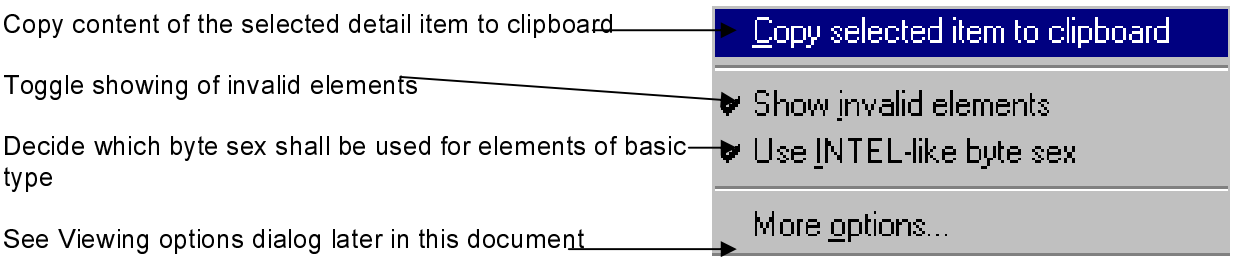
3.3.1.4 Context Menus

Inside the two main windows you may open context menus:

- in the upper window:

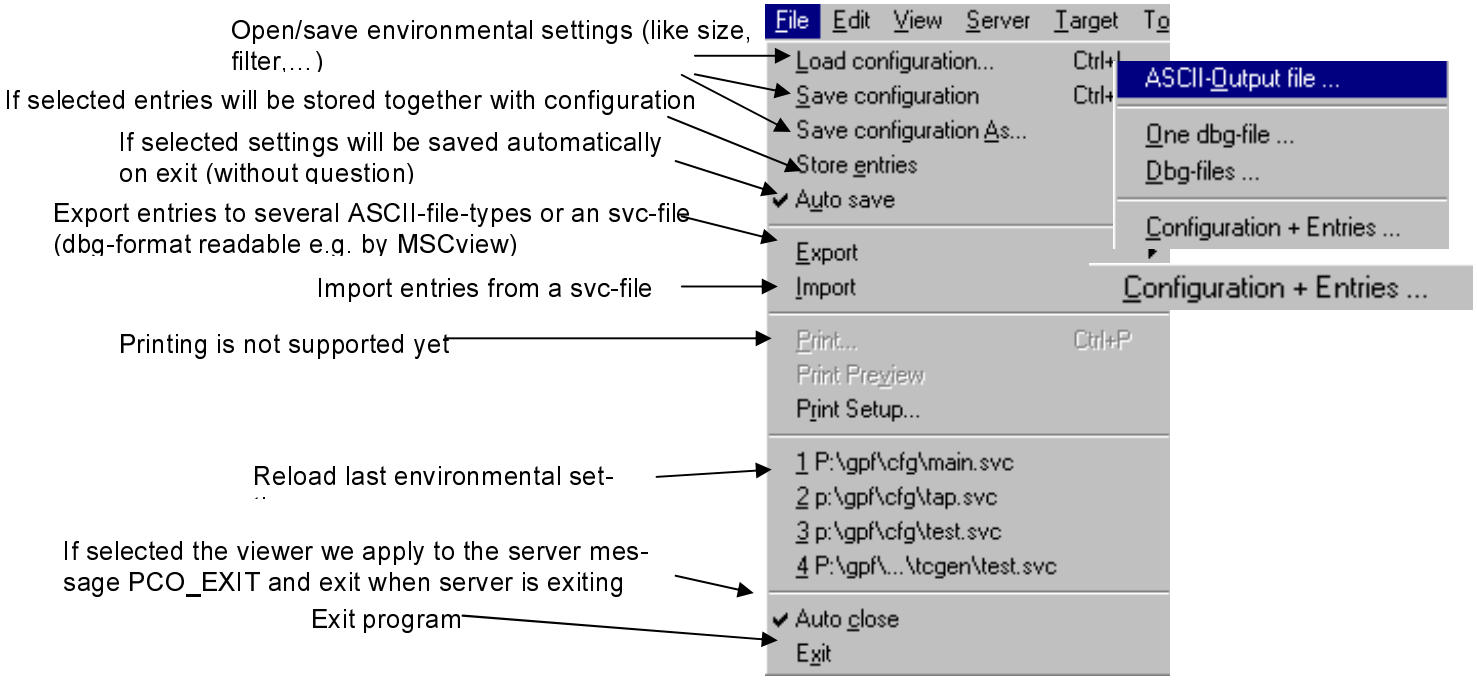


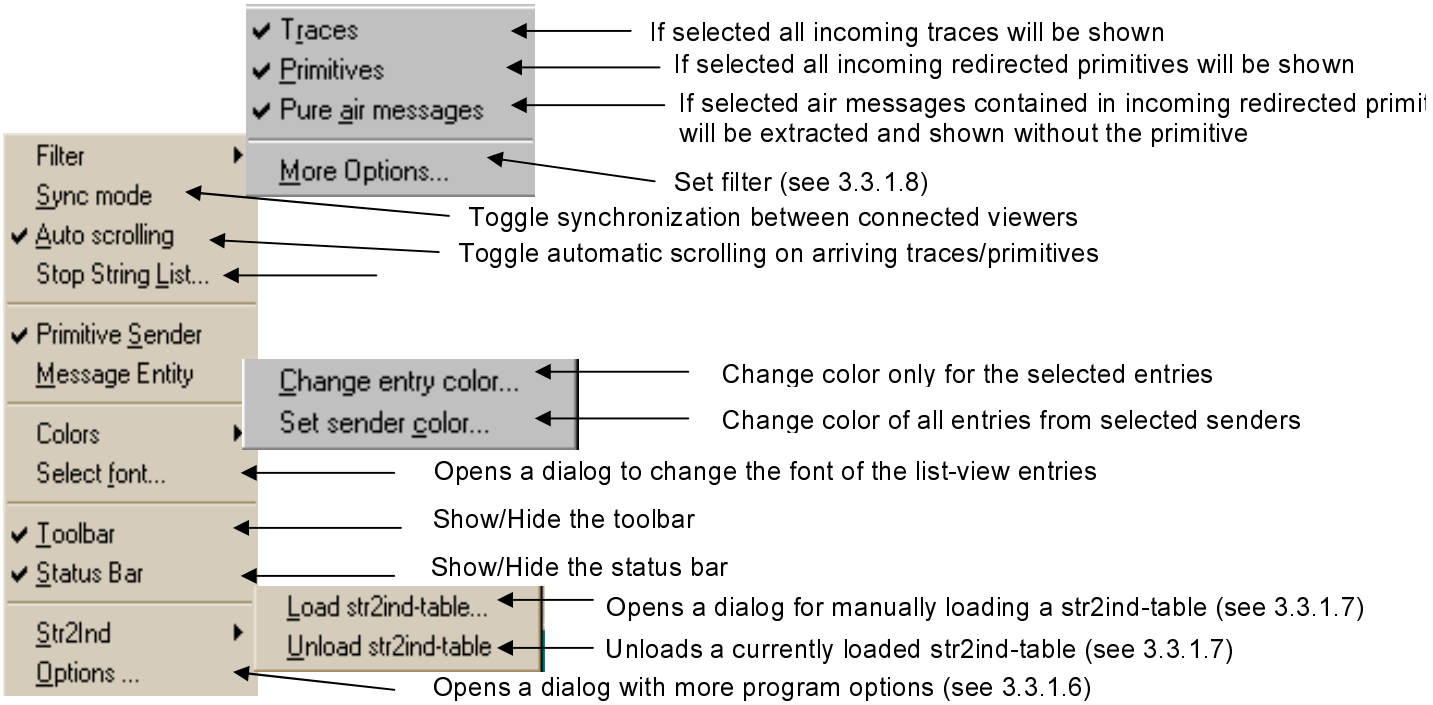
- in the lower window:

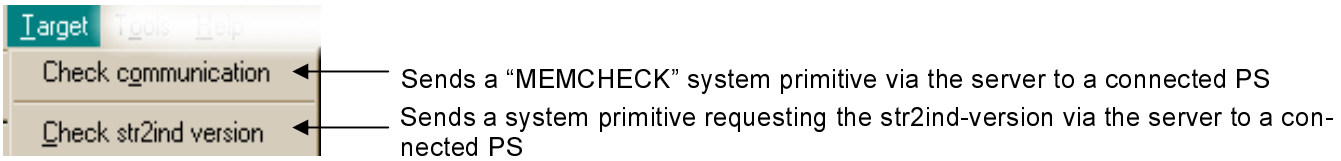


3.3.1.5 The Menu

The application menu offers some more options described below:

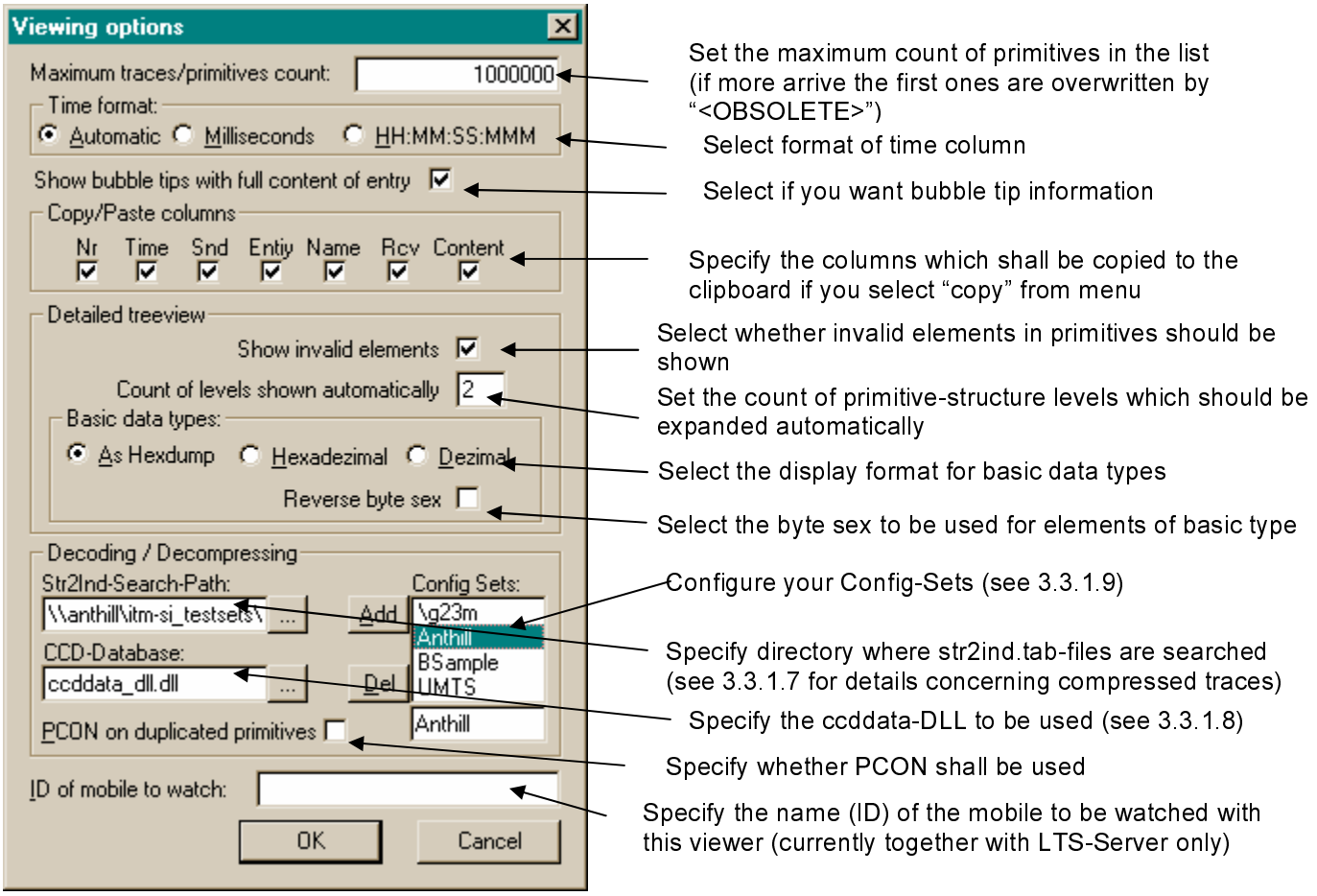






Via the <tool>-menu other tools like MSCView may be invoked on the currently selected entries (or on all, if none is selected).

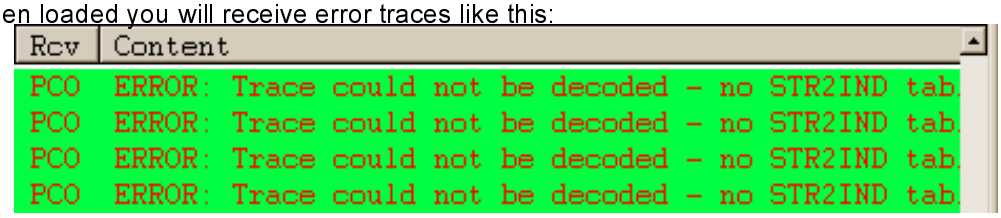
3.3.1.6 The Options-Dialog



3.3.1.7 Compressed traces and str2ind

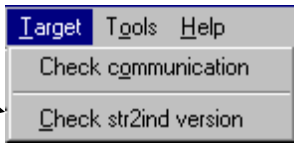
With version 1.3.2 of the Condat GPRS protocol stack so called "compressed tracing" has been introduced. To decompress the traces PCO needs a table (str2ind.tab) which is generated each time a new protocol stack is build. For detailed information see [STR2IND].

The standard viewer searches for *.tab-files in the directory specified in the "View/options"-Dialog and all sub folders. (This directory is stored in the PCO-ini file in the [Viewers]-section)
If no table has been loaded you will receive error traces like this:



Furthermore the status field in the tool bar will state the fact:

Each time the protocol stack is reset it will send a str2ind-version-number automatically, so that PCO can try to load a table. Also after receiving some compressed traces without having a table loaded PCO will request that number from protocol stack. To manually request the version number you may select "Check str2ind version" from the menu:



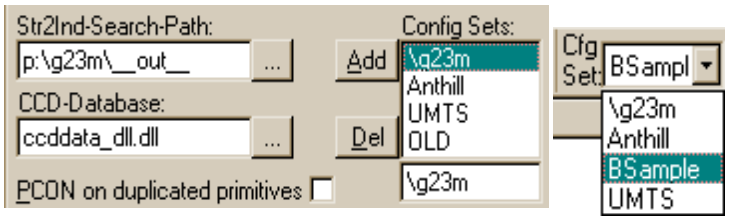
To manually load a specific str2ind-table use the entry in the <view>-menu. This should only be done if you are actually certain, that the tab-file is matching the current protocol stack !!

3.3.1.8 Selecting the CCDDATA-library

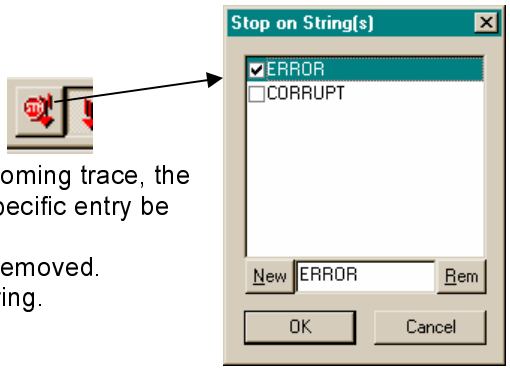
In the options dialog (see 3.3.1.6) you can select a specific ccddata-DLL which shall be used for primitive interpretation and air message decoding. The selection will be stored in the pco.ini-file and all other connected viewers will be informed – so you have to select it only in on viewer ! The ccddata-DLL is usually build by the makcdg-script and delivered together with a PS.

3.3.1.9 Config-Sets

A config set contains a str2ind-path, a ccddata-DLL and a PCON-setting. With the “Add” and “Del” config sets can be added and removed. In the edit-box below the list the name of the selected set can be edited. From the toolbar you may quickly change between configured config sets and so e.g. between str2ind paths to be used.



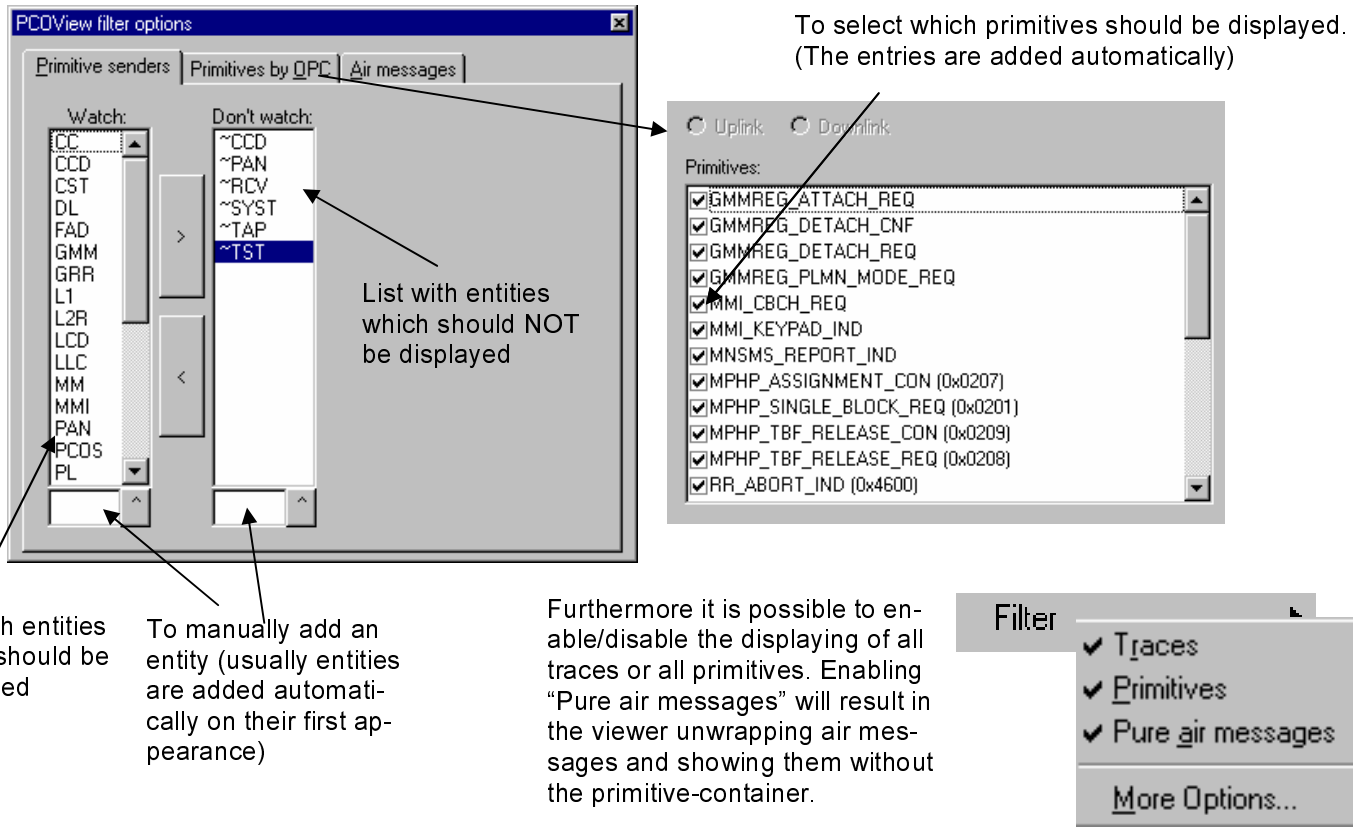
3.3.1.10 The STOP-String-List



In this list you can specify strings which will be searched in the incoming entry-flow .
Whenever such a string is found (case is irrelevant) in an incoming trace, the automatic scrolling of the entry-list will be stopped and the specific entry be selected. (Per default the string “ERROR” is selected.)
Via the “New” and “Rem” buttons strings can be added and removed.
The edit-box can be used to modify the currently selected string.

3.3.1.11 Configuring the filter

Currently the CCD-supported version of the standard viewer provides to kinds of filters: one depending on the sender of a primitive/trace and one depending on an evtl. contained OPC. (The CCD-free version, pco_miniview.exe, provides only the sender-filter). The following pictures describe the functionalities of these:



To select which primitives should be displayed. (The entries are added automatically)

List with entities which should be displayed

To manually add an entity (usually entities are added automatically on their first appearance)

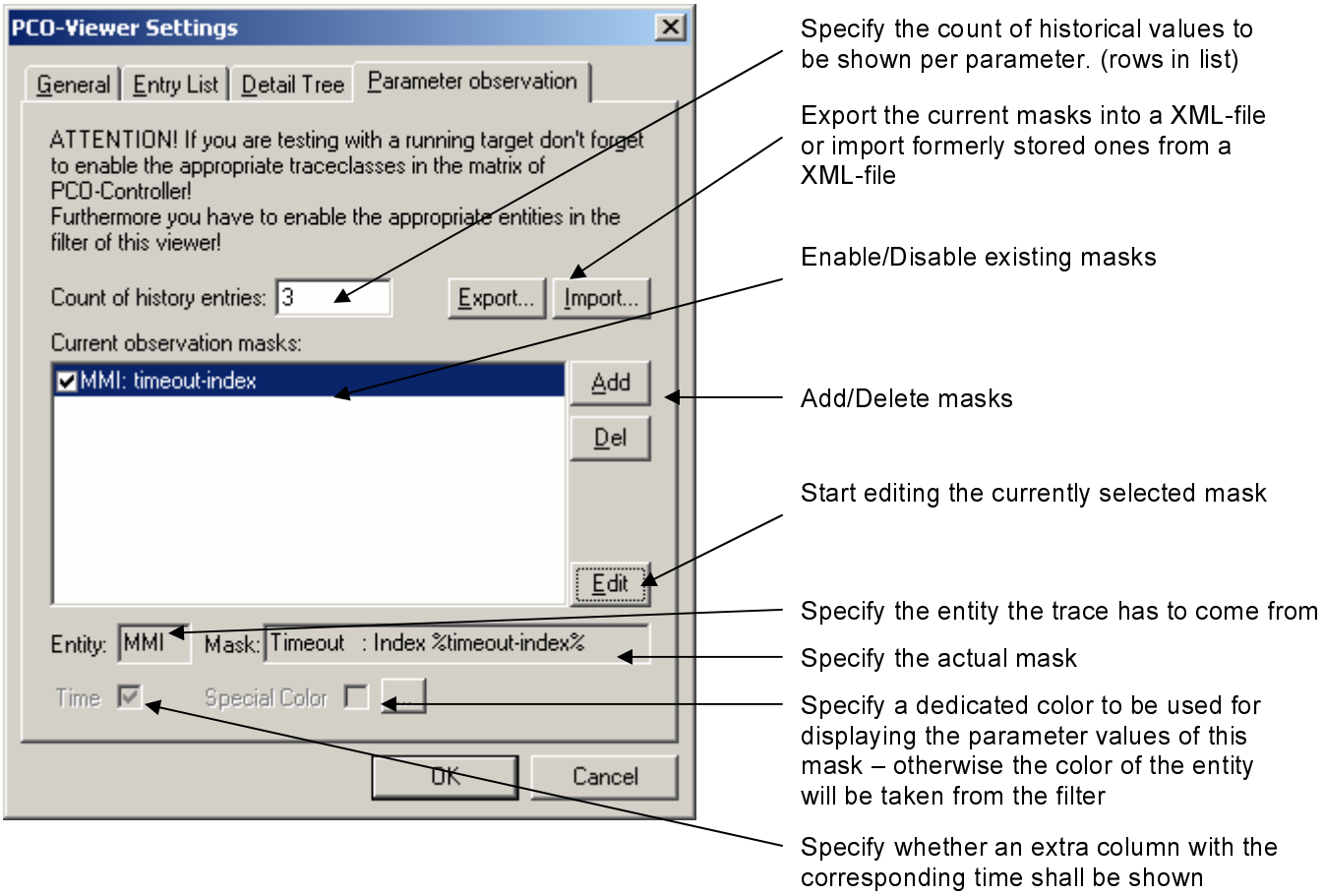
Furthermore it is possible to enable/disable the displaying of all traces or all primitives. Enabling “Pure air messages” will result in the viewer unwrapping air messages and showing them without the primitive-container.

3.3.1.12 Parameter Observation

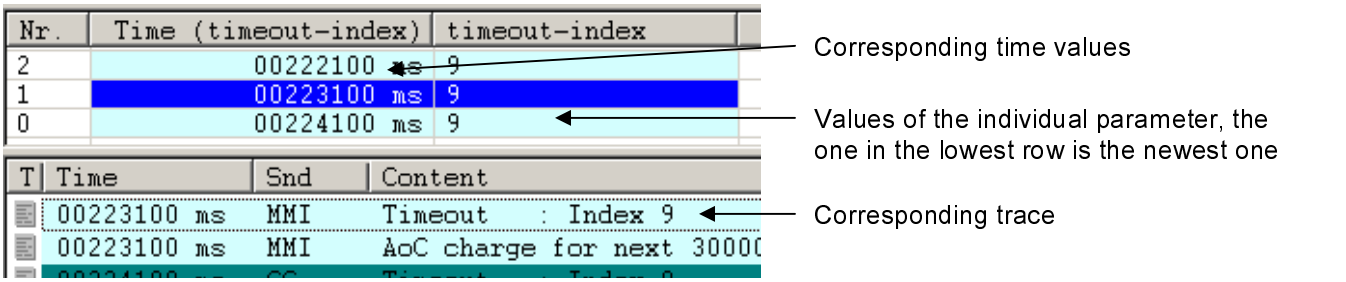
The parameter observation feature of the Standard Viewer allows the user to specify various masks which will be applied on incoming trace-strings to retrieve the current values of dedicated PS or entity parameters. Such masks can be specified in the appropriate sheet of the options-dialog, accessible via menu <view><options> or via the context menu of the parameter-observation list (the first of the three sub windows of the viewer.

A mask contains characters which are compared with the ones in the traces and strings enclosed by '%'-s which are placeholders for the parameters to be read, but they also define the name of a parameter. As an example the mask "Timeout : Index %timeout-index%" will match traces like "Timeout : Index 3" and in the latter case "3" will be recognized as the current value of parameter "timeout-index". Former values will be shifted up in the list and the new one is inserted.
Matching is done until the end of the mask, so if a trace contains additional characters these are ignored. On the other hand traces which are shorter then the mask will not match.

The following figure explains the functionalities of the options-dialog sheet:



The next figure explains the output fields in the parameter-observation list:



3.3.2 Other viewers

Beside the standard MFC-viewer described above several viewers have been developed for specific needs.

Pco_dump.exe is a simple command line viewer dumping all received traces and primitives to stdout.

The test case generator TCGen (see [TCGEN]) is also internally using the viewer concept to request PCO logfiles.

3.3.3 Deriving a new viewer

This chapter is intended to give potential developers of new PCO viewers a general description of how to do this. See Figure 1 for a first overview.

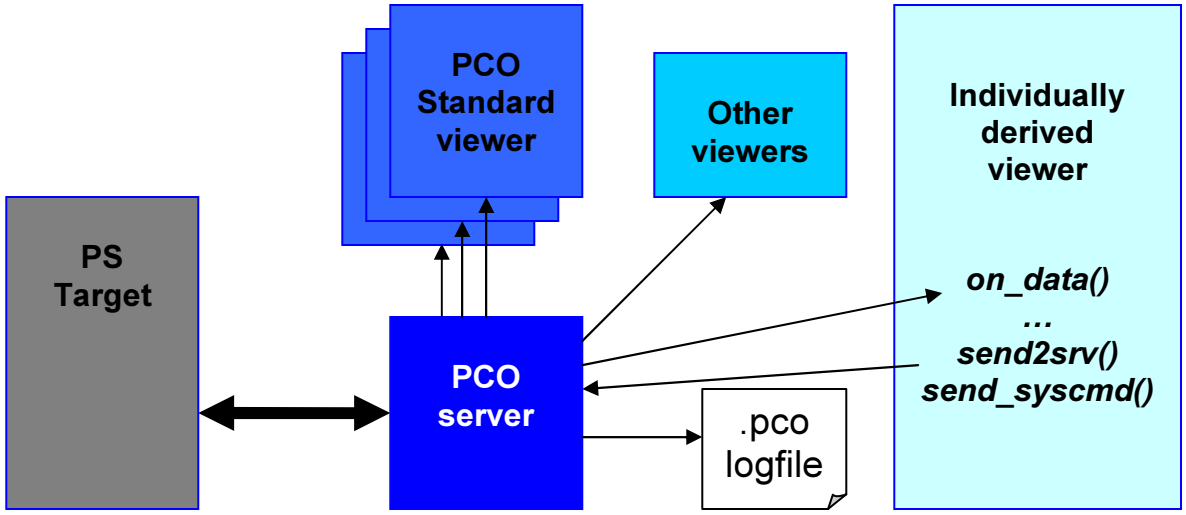


Figure 1 – Communication overview between PCO server and viewers

Of course you can look into [PCO2_D] and implement your viewer from scratch using CMS and the PCO communication constants. But it is more easy to apply the object oriented paradigm and use a dedicated PCOView base class.
There are a lot possibilities: basing on PCOView_templ, PCOView_core or PCOView_frameSupp.
Independent which base will be used "..\INC" has to be added to the include path.
For the detailed description of the PCO_XXX control messages see "..\INC\pco_const.h".
IMPORTANT: You have to use compiler switch "/Zp1" to set the alignment to 1 byte for all derived viewers !

To get a first impression of how you could start there exists a so-called DemoViewer whose sources are delivered together with PCO.

3.3.3.1 Using PCOview_templ

If you already have an application with CMS-queues and only want to connect to a PCO server to retrieve traces and redirected primitives you should derive from the class PCOview_templ defined in "..\INC\pco_view_templ.h".

While you have to provide the names of your queues (one for control messages, one for the trace etc.) as parameters of the constructor this class implements the following functions:

int connect(void):

- PURPOSE: tries to connect with server by sending PCO_CONNECT with the queue names
- RETURNS: 0 .. success
  -1 .. Server not found
  -2 .. error while contacting Server

int subscribe(const char mobileId):*

- PURPOSE : tries to subscribe for live data from server
- PARAMS: mobileId .. name of mobile to receive live data from (may be empty)
- RETURNS: 0 .. success
  -1 .. Server not found
  -2 .. error while contacting Server

int unsubscribe(void):

- PURPOSE : tries to unsubscribe from livedata stream from server
- RETURNS: 0 .. success
  -1 .. Server not found
  -2 .. error while contacting Server

int disconnect(void):

- PURPOSE: tries to disconnect from server by sending PCO_DISCONNECT
- RETURNS: 0 .. success
  -1 .. Server not found
  -2 .. error while contacting Server

int send2srv(void buf, U16 size, U16 id):*

- PURPOSE : tries to send a data buffer to the server
- PARAMS: buf ... pointer to buffer
  size .. size of buffer
  id ... message id
- RETURNS: 0 .. success
  -1 .. Server not found
  -2 .. error while contacting Server

*virtual int on_connected(const void *buf,const char* sender):*

- PURPOSE : reaction to PCO_CONNECTED from server
  can be overwritten if needed
- PARAMS: buf .. data containing server type
  sender .. server queue name
- RETURNS: 0 .. server type supported
  -1 .. server type not supported

int get_logdata(ULONG begin, ULONG end):

- PURPOSE: contacts server to request logged data by sending PCO_GET_LOGFILE_DATA with the params
  -> if the server is in logfile-mode it will send the requested data (traces etc.) to the primitive-queue
- PARAMS: begin, end .. time area requested (in milliseconds)
- RETURNS: 0 .. success
  -1 .. Server not found
  -2 .. error while contacting Server

int open_logfile(const char fname):*

- PURPOSE : contacts server to request logged data from a specified logfile
- PARAMS: fname .. name of logfile (full path)
- RETURNS: 0 .. success
  -1 .. Server not found
  -2 .. error while contacting Server

int set_filter(const char list)*:

- PURPOSE: contacts server to change filter settings by sending PCO_SET_FILTER
  -> the server will apply to the new filter when sending the next primitive etc.
- PARAMS: list .. '\0'-separated list of names of entitities
  (terminated by "\0\0")
  primitives/traces from entities in -list-
  should not be forwarded
- RETURNS: 0 .. success
  -1 .. Server not found
  -2 .. error while contacting Server

void set_srv_name(const char sname):*

- PURPOSE : change the name of the server queue to be used
- PARAMS: sname ... new name of server queue

Note: when using PCOview_templ you have to provide one or more threads which read the data from the cms-queues.

Furthermore you have to link your viewer application with the following libraries:

- "..\LIB\WIN32\pco_view.lib" … contains the basic viewer classes and its functions and also various FRAME related libs

The debug versions of all libraries can be found in an appropriate sub directory "debug". They are named the same.

As could be suspected by the usage of import libraries you'll have to provide several DLLs together with your viewer application, too:

- "..\BIN\cms.dll" … contains VCMS (Virtual Condat Multitasking System, see [CMS])
- "..\BIN\frame.dll", "..\BIN\misc.dll", "..\BIN\tif.dll" ... FRAME related DLLs

For the debug versions of the DLLs the same rules apply as for the libraries above.

3.3.3.2 Using PCOview_core

If you want to develop a new viewer for PCO with general access to data forwarded by the PCO server the easiest way is to derive a class from PCOView_core defined in "..\INC\pco_view_core.h". It will provide you with the basic functionality a PCO viewer needs. Through its base class PCOview_templ it has member functions to connect/disconnect to/from a PCO server, request logged primitives and traces and set filters (see 3.3.3.1). Furthermore two threads will be started which read from the internally managed cms-queues. Control messages will result in a call to *dispatch_message()* which can be specialised in your derived class.

All traces and redirected primitives are pre-interpreted by *interpret_message()* and, if this succeeded, forwarded to *on_data()*. Both functions are pure virtual and have to be provided by your viewer class.

Each PCO viewer derived from PCOView_core expects an ini-file which name has to be provided to the constructor. If an empty string is used, a default ini-file will be used if existing.

Here are detailed descriptions of the mentioned functions:

virtual int dispatch_message(void buf, U16 size, U16 id, const char* sender):*

- PURPOSE : parses a PCO control message
- PARAMS: buf ... the data
  size .. size of buf
  id ... id of the message
  sender .. queue name of sender
- RETURNS: 0 .. if message has been handled
  -1 .. otherwise

virtual int interpret_message(void buffer, U16 bufsize, void* &data, U16 &size, ULONG &id, U32 &time, char* &sender):*

- PURPOSE : here interpretation of received raw data takes place
  (has to be implemented by derived classes !)

- PARAMS: buffer .. raw data to be interpretated
  bufsize .. size of buffer
  data .. actual data
  size .. size of data
  id .. id of data message
  time .. time stamp in ms
  sender .. name of sender

- RETURNS: 0 .. success
  -1 .. interpretation was not possible

virtual void on_data(void data, U16 size, ULONG id, ULONG index, ULONG ttype, U32 time, const char* sender, const char* receiver):*

- PURPOSE : here reaction to received data takes place
  (has to be implemented by derived classes !)

- PARAMS: data .. the data
  size .. size of data
  id .. id of data message
  index .. index of data message (e.g. in logfile) ... 0 means no index!!
  ttype .. type of time stamp - see PCO_TTYPE_XXX constants
  time .. time stamp
  sender .. name of sender
  receiver.. name of original receiver

int propagate_inichange():

- PURPOSE : saves ini-file and sends an information about important ini-file changes to the server, which will propagate it to all connected viewers

- RETURNS: 0 .. success
  -1 .. Server not found
  -2 .. error while contacting Server

virtual void self_trace(const char trace):*

- PURPOSE : adds a new trace string to the queue of this viewer
  (has to be implemented by derived classes !)

- PARAMS: trace .. trace string

Note: when using PCOview_core you don't have to provide any thread which reads data from the cms-queues.

Of course you have to link your viewer application with some libraries:

- "..\LIB\WIN32\pco_view.lib" … contains the basic viewer classes and its functions
  various FRAME related libs

The debug versions of all libraries can be found in an appropriate sub directory "debug". They are named the same.

As could be suspected by the usage of import libraries you'll have to provide several DLLs together with your viewer application, too:

- "..\BIN\cms.dll" … contains VCMS (Virtual Condat Multitasking System, see [CMS])
- "..\BIN\frame.dll", "..\BIN\misc.dll", "..\BIN\tif.dll" ... FRAME related DLLs

For the debug versions of the DLLs the same rules apply as for the libraries above.

3.3.3.3 Using PCOView_frameSupp

As stated in 3.3.3.2 when using PCOView_core you have to provide all the interpretation functionality of incoming data by your new class. With PCOView_frameSupp you get a heir of PCOView_core providing an implementation of *interpret_message()* which currently supports data formatted as a SYST_MESSAGE (e.g. coming from the FRAME testinterface and forwarded by the PCO server) or as an ASCII-stream (e.g. coming other the serial connection and forwarded by the LTS server). *on_connected()* will return 0 if either of the supported servers is used. You may get the server type via *srv_type().*

Furthermore *interpret_message()* applies a conversion from indices to strings if necessary and if an ind2str table has been loaded.

decode_tracestring() can be used in *on_data()*, which still has to be implemented by the new viewer, to translate OPCs in traces to primitive names.

Like with the PCOView_core derived viewers an ini-file is expected to be provided to the constructor. If an empty string is used, a default ini-file will be used if existing.

Here are detailed descriptions of the mentioned functions:

int decode_tracestring(const char instr, char* outstr, U16 size):*

- PURPOSE : tries to decode an OPC in the tracestring ($<OPC>)
- PARAMS: instr .. original tracestring
  size .. max size for outstr
- RETURNS: 0 .. success (outstr contains new tracestring)
  -1 .. no success while decoding
  -2 .. no decoding necessary

int send_syscmd(const char receiver, const char* cmd):*

- PURPOSE : tries to send a FRAME system command to the server
  who will forward it to a connected protocol stack
- PARAMS: receiver ... receiver of the command
  cmd ... the actual command (e.g. "TRACECLASS FF")
- RETURNS: 0 .. success
  -1 .. Server not found
  -2 .. error while contacting Server

Note: when using PCOView_frameSupp you don't have to provide any thread which reads data from the cms-queues.

Of course you have to link your viewer application with some libraries:

- "..\LIB\WIN32\pco_view.lib" … contains the basic viewer classes and its functions
  various FRAME related libs

If CCD data should be interpreted you furthermore need these:

- "..\LIB\WIN32\ccdedit.lib" … contains functions for access to the "ccddata_dll.dll"
- "..\LIB\WIN32\ccddata_loadlib" … import library for the "ccddata_load.dll"

The debug versions of all libraries can be found in an appropriate sub directory "debug". They are named the same except for "misc.lib" whose debug version is "misc.lib".

As could be suspected by the usage of import libraries you'll have to provide several DLLs together with your viewer application, too:

- "..\BIN\cms.dll" … contains VCMS (Virtual Condat Multitasking System, see [CMS])
- "..\BIN\frame.dll", "..\BIN\misc.dll", "..\BIN\tif.dll" ... FRAME related DLLs

If CCD data should be interpreted you furthermore need these:

- "..\BIN\ccddata_load.dll" … loader for ccddata_dll.dll
- ccddata_dll.dll … has to be rebuild after each call of makcdg

For the debug versions of the DLLs the same rules apply as for the libraries above.

4 Known problems and future tasks

This paragraph is meant to show which bugs are already found (but not removed yet) and to provide an impression of future plans concerning this product.

4.1 Known bugs

-

4.2 „Soon implemented“

-

4.3 „Nice to have“

- Online-Help

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/>