Technical Document

GSM PROTOCOL STACK

G23

PCO2 – TRACING ENVIRONMENT

DEVELOPER DESCRIPTION

Document Number:	06-03-35-SLL-006
Version:	0.6
Status:	Draft
Approval Authority:	
Creation Date:	2000-Dec-15
Last changed:	2015-Mar-08 by Ronny Kiessling
File Name:	Pco_description.doc

Copyright © 2003 Texas Instruments
Texas Instruments Proprietary Information – Internal Data
technical_document_20030404.dot

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.

No part of this document may be reproduced or transmitted in any form or by any means, electronically or mechanically, including photocopying and recording, for any purpose without the express written permission of TI.

Table of Contents

0 Document Control

0.1 Change History

Date	Changed by	Approved by	Version	Status	Notes
2000-Dec-15	RK et al.		0.1	Being Processed	1
2001-May-28	RK		0.2	Being Processed	2
2001-Nov-15	RK		0.3	Being Processed	3
2003-May-21	XINTEGRA		0.4	Draft	
2003-Aug-18	RK		0.5	Draft	4
2003-Sep-17	RK		0.6	Draft	3

Notes:

1. Initial version
2. First official version
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_UG]	06-03-35-UDO, PCO2 – Tracing Environment (pco_userguide.doc)
[CRULES]	8415.100.00.103, July 18, 2000, C Coding Standard (gpf_memo_crules.doc)
[CMS]	8309.201.97.002, ?? ??, 19??, CMS 3 - Handbuch (cms_man.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:

PCO is a tool that finally provides an interface for tracking all kind of traces and primitives. Originally it was designed as an CONDAT-internal tool only but there are already some customers using the "old" PCO as well. The old 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 (or more) view(s). Several requirements have been designated (see 1.1 and 1.2)

Until the end of May 2000 SH worked on a stand-alone test-version (called pco2000, see file:\\U:\sh\pco2000) which containes a server, a controller and a minimal view. The components communicate with a message format that is related to the system messages the FRAME uses.

In August 2000 RK started to adapt pco2000 and develop the new PCO with the given requirements.

Currently there are only Win32-versions (GUI supported or command line based) available, but since the core sources are VCMS-based an adaptation to, e.g. Linux, should be easy.

This documentation is dedicated to interested developers. For user/customer specific documentation see [PCO2_UG].

1.1 Requirements - under construction -

The following requirements have been designated to the new PCO:

- Ablauf soll AUF JEDEN FALL ohne GUI möglich sein. (EXTREM WICHTIG!). Keine GUI Einstellungen notwendig.

Grundidee

Um die Erweiterbarkeit zu gewährleisten wird PCO mit verschiedenen Prozessen implementiert. Der erste Prozess (ab jetzt **PCOServer**) hat keine GUI und ist „nur" dafür zuständig, dass der ankommende Nachrichten Datenstrom in eine Datei gestreamt wird, sämtliche Einstellungen lassen sich über Kommandozeilenparameter, über ein ini File oder durch CMS Nachrichten.
Zusätzlich kann der PCOServer ein vorher gespeichertes Testsession - Logfile laden.
Für jede Grafikanzeige (Viewer) wird ein weiterer Prozess implementiert, es ist hierbei völlig egal, ob der Prozess Daten auf den Bildschirm anzeigt, sie auf Port 80 als Webserver anbietet, per Sprachausgabe ausgibt oder mit dem Palm Pilot Synchronisiert. Ein Viewer muss sich bei PCOServer anmelden und bekommt dann die ankommenden Daten über CMS gestreamt oder einen Ausschnitt aus einem geladenem Logfile, je nach Anfrage. An einem PCO Server können mehrere Viewer laufen um z.B. verschieden gefilterte Outputs zu erlauben.
Eine weitere optionale Client Komponente ist ein Controller, ein Controller ist das GUI Interface des PCO Servers, es kann immer nur einen Controller pro Server geben. Der PCO Server läuft allerdings auch völlig ohne GUI, Kommandozeilenbasiert.

PCOServer
PCOServer basiert auf CMS und ist während des Tests für das Schreiben von einem Logfile verantwortlich, nach dem Test kann PCOServer Logfiles laden und diese dann anzeigen.
Eine Steuerung ist über Kommandozeile, wie auch über CMS Nachrichten möglich.
Es können mit PCOS_FORK_ mehrere Instanzen vom PCO Server erzeugt werden, um mehrere Files zu Laden (Muss aber nicht). Die erste Instanz heißt immer PCOS0.
Einstellungen
Folgende Einstellungen lassen sich bei PCO Server über ein ini File oder Kommandozeilenparameter machen.

- Dateiname in welchen gestreamt wird
- Dateiname welcher geladen wird
- ini File, welches bei start geladen wird (nur Kommandozeilenparameter)
- Maximale Logfilegrösse, bis Testsessionabbruch

- Einstellung ob während des Ableufs CMS Kommados zur Steuerung akzeptiert werden
- Einstellung des Routings im Frame

•

1.2 Features - under construction -

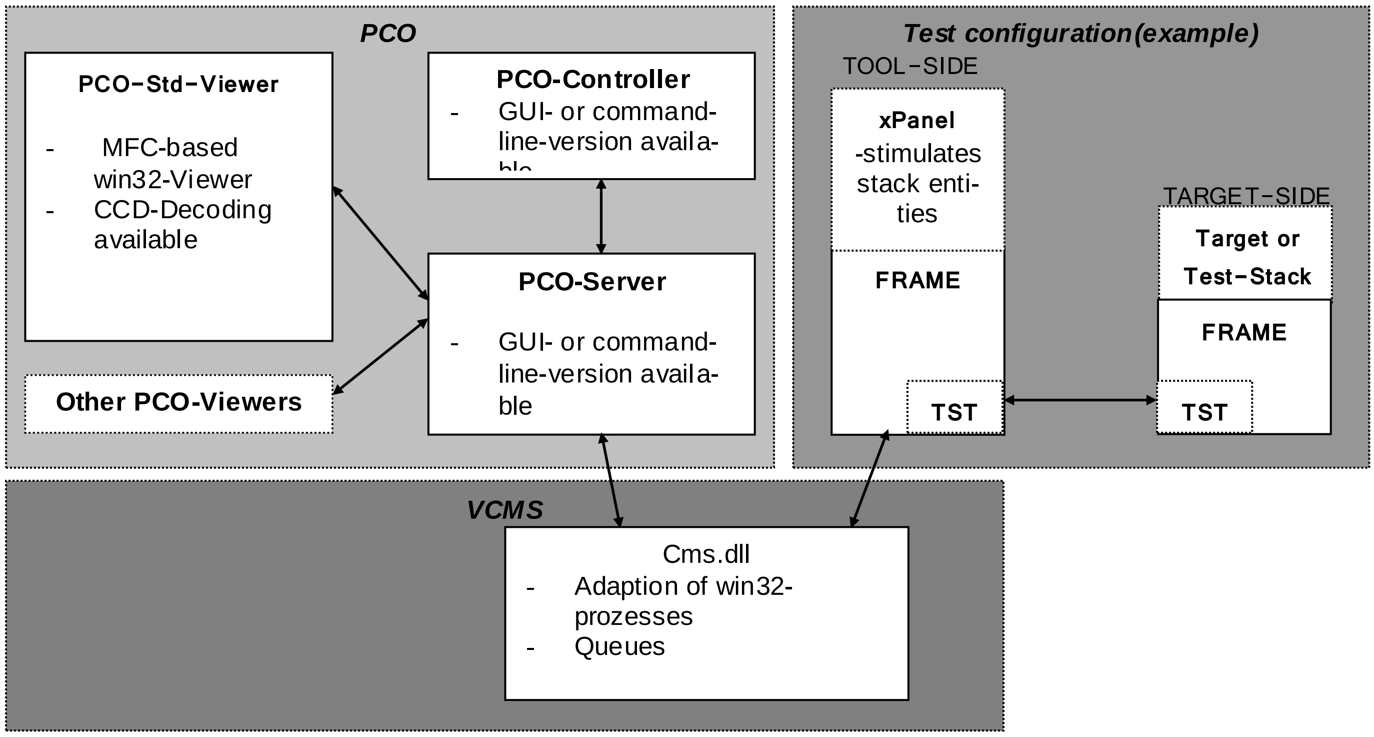
- wichtig ist vor allem **die Auswertung der Logfiles** nach dem Test. Der eigentliche **Test braucht keinen Komfort** Bookmarks, Zeitleiste und Trace - Notizen als Auswertungshilfen, sowie ein Logfile Verwaltungs System.
- Weg von dem je Entity ein Fenster und eine Datei Konzept, ein Gesamtfenster welches gefiltert wird ist Besser. Logfile nur noch eine Datei.
- Dafür Typographische Unterscheidungsmerkmale zwischen verschiedenen Nachrichten zur optischen Unterscheidung. (Syntax Highlighting)

2 Solution

2.1 General Procedure

As described before PCO consists of several components communicating via the VCMS interface (CMS-Queues). The PCO-Server is the only part which connects to the FRAME on tool side. It can be controlled by the PCO-Controller and distributes all primitives (coming from the Stack or read from a logged session-file) to connected viewers.

The following picture shows the dependencies between the components:



2.1.1 Environment

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-files are available to the system. In the Condat development directory structure you can find them in „<View>/GPF/Bin" :

- CMS.dll
- Ccddata_dll.dll (if you want to use CCD-support)

So just make sure „<View>/GPF/Bin" is in your PATH-variable.
Furthermore you have to rebuild the "Ccddata_dll.dll" every time you call "makcdg". You can do that by calling "gnumake –f ccddata.mk" in „<View>/GPF/CCD". (call "gnumake -f ccddata.mk help" for more options)

2.1.2 Build process

To build the PCO components you can either use gnumake with the provided makefile or look into the MSDev sub directory for predefined project/workspace files.

The makefile options can be shown by running "gnumake help" in the PCO directory. Their meanings are as follows:

gnumake [srv|ctrl|views|all [CCD={1|0}]|bat|ini] [DEBUG={1|0}] [GUI={1|0}] [EXPORT={1|0}]

- build all (default) or one dedicated component (srv..server, ctrl..controller, views..viewers, bat..batch-files, ini..ini-files)
- CCD ... enable/disable CCD support
- DEBUG ... enable/disable debug version
- GUI ... build command line or GUI version
- EXPORT ... if 1 all binaries, batch- and ini-files are copied to the global bin-directory (GPF/BIN or GPF/BIN/debug)

gnumake {clean|clean_srv|clean_ctrl|clean_views} [DEBUG={1|0}] [CCD={1|0}] [GUI={1|0}]

- clean all or one dedicated component
- DEBUG, CCD, GUI see above

gnumake checkin [CICMT=<checkin comment>] [DEBUG={1|0}] [CCD={1|0}] [GUI={1|0}]

- check in all files of PCO with checkin command CICMT
- DEBUG, CCD, GUI see above

gnumake label LABEL=<label type> [FLOAT={1|0}] [DEBUG={1|0}] [CCD={1|0}] [GUI={1|0}]

- Label all files of PCO with label LABEL
- FLOAT ... if 1 label will be moved if already on older version
- DEBUG, CCD, GUI see above

Temporary object files are stored in *pco/obj* and its sub directories. After successful build the binaries can be found in *pco/bin* or *pco/bin/debug.*

To find out more about the specific sources see 2.2.

2.1.3 Programming/Naming rules

Every effort has been made to apply to the rules defined in [CRULES]. Anyway some code which has been reused/modified may not fully comply to it.

In addition to the global guideline some "internal" rules have been defined and applied:

- Static module variables or member variables of classes are prefixed by "m_"

2.1.4 Command line parameters and ini-files

From the command line you can specify some environmental options for the pco-components.
See 2.2.1.4, 2.2.2.3 and 2.2.3.1.3 for component specific informations.

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 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]

- QueueSize .. size of the queue for traces/primitives in bytes (e.g. 1400)

An overview of the available component specific parameters is given in the sections 2.2.1.4 and 2.2.2.4.
You can find examples in the standard pco.ini file.

2.2 Detailed Descriptions

2.2.1 PCO-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 which can be linked to the xPanel (see [XPAN]), the TAP or be 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).
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 will change its colors and show more informations in a dialog window.

Like all PCO components the server uses CMS-queues for communications. The one named "PCOS" is the queue to receive control messages from the controller or any viewer. The "PCO"-queue is provided for connection to the testinterface-entity. Here all traces and redirected primitives will be received. For more informations about communication see 3.3.

Currently only one server can run at a time.

2.2.1.1 States

The several states the server can be set to are:

- **PCO_STOPPED**
  All primitives from testinterface are routed to connected clients. (initial state)
- **PCO_RUNNING**
  All primitives from testinterface are forwarded to connected clients and stored to a session log file.
- **PCO_LOGFILE**
  All primitives from testinterface are routed to connected clients. Further more on demand from a client the selected session file will be opened and its content send to this client. (Some changes will occur here in next version)

They are defined in *pco_const.h* - *T_PCOSTATUS*.
It is not possible to reach PCO_RUNNING from PCO_LOGFILE without coming over PCO_STOPPED.

2.2.1.2 Classes and functions - under construction -

2.2.1.3 Sources - under construction -

The C/C++ sources of PCO server can be found in the PCO directory in *src* and *src/srv.* They are in detail:

• *IniFile.cpp*	
• *pco.ini*	default ini file (see 2.2.1.4)
• *pco2.bat*	batch file for starting default set of PCO components
• *pco_util.cpp*	common utility functions
• *syst_prim.cpp*	
• *srv/pco_client.cpp*	
• *srv/pco_clientlist.cpp*	

- *srv/pco_srv_core.cpp*
- *srv/pco_srv_framesupp.cpp*
- *srv/pcod.cpp*
- *srv/win32/Led.cpp*
- *srv/win32/StdAfx.cpp*
- *srv/win32/pco_srv.cpp*
- *srv/win32/pco_srv.rc*
- *srv/win32/pco_srvDlg.cpp*
- *srv/win32/pco_srv_framesupp.cpp*
- *srv/win32/resource.h*
- *srv/win32/systray_icon.cpp*

Include files used by the server are saved in *inc* and *inc/srv.* Their content is described below:

- *IniFile.h*
- *pco_const.h*
- *pco_util.h*
- *syst_prim.h*
- *srv/pco_client.h*
- *srv/pco_clientlist.h*
- *srv/pco_srv_core.h*
- *srv/pco_srv_framesupp.h*
- *srv/win32/Led.h*
- *srv/win32/StdAfx.h*
- *srv/win32/content.txt*
- *srv/win32/pco_srv.h*
- *srv/win32/pco_srvDlg.h*
- *srv/win32/pco_srv_framesupp.h*
- *srv/win32/systray_icon.h*

The GUI-version uses some resources situated in *res/srv*:

- *green.ico*
- *green2.ico*
- *icon1.ico*
- *leds.bmp*
- *pco_srv.ico*
- *pco_srv.rc2*
- *red_off1.ico*
- *yellow.ico*
- *yellow_o.ico*

2.2.1.4 Command line parameters

On the command line of the GUI- and NON-GUI-server you may specify the following options which can be listed with "pco_srv –h" or "pcod –h" as well:

- -i <ini-file> … use specified ini-file

2.2.1.5 Ini-files settings

In the pco.ini file (see 2.1.4) 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 to the testsessions stored by the server (shouldn't be on a ClearCase-View because of performance reasons)

2.2.2 PCO-Controller

The controller is the tool to manipulate the state of the server. It could be integrated in a viewer but since it occupies a CMS-queue only one instance can be started at one time. In the current implementation there is a command line version with basic functionalities and a GUI-version which can e.g. send Config-primitives to the server. If the latter understands such messages it for-ward the primitives to the testinterface.

The CMS-queue used for receiving control messages from server is named "PCOC".

2.2.2.1 Classes and functions - under construction -

2.2.2.2 Sources - under construction -

The C/C++ sources of PCO controller can be found in the PCO directory in *src* and *src/ctrl.* They are in detail:

- *IniFile.cpp*
- *pco.ini* default ini file (see 2.2.2.4)
- *pco2.bat* batch file for starting default set of PCO components
- *pco_util.cpp* common utility functions
- *syst_prim.cpp*
- *ctrl/StdAfx.cpp*
- *ctrl/pco_ctrl.cpp*
- *ctrl/pco_ctrl.rc*
- *ctrl/pco_ctrlDlg.cpp*
- *ctrl/pco_ctrl_core.cpp*
- *ctrl/pcoc.cpp*
- *ctrl/resource.h*

Include files used by the controller are saved in *inc* and *inc/ctrl.* Their content is described below:

- *IniFile.h*
- *pco_const.h*
- *pco_util.h*
- *syst_prim.h*
- *ctrl/StdAfx.h*
- *ctrl/pco_ctrl.h*
- *ctrl/pco_ctrlDlg.h*
- *ctrl/pco_ctrl_core.h*

The GUI-version uses some resources situated in *res/ctrl*:

- *pco_ctrl.ico*
- *pco_ctrl.rc2*

2.2.2.3 Command line parameters

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

- start <testname> … start a test session named –testname-

- stop ... stop a currently running test session
- open <logfile name> ... open a logfile (stored test session)
- send <receiver> <text> ... send a CONFIG-primitive –text- to –receiver- (Entity name)
- close ... close currently opened logfile
- exit ... exit a currently running server
- info [<logfile name>] ... print info about current server state/ specified logfile

2.2.2.4 Ini-files settings

In the pco.ini file (see 2.1.4) you can specify several controller settings.

[Controller]

- PrimFile .. file with predefined CONFIG-primitives for Controller (usual "view.txt", see [PCO2_UG])
- 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 2.2.2.5)
- StackConfigFile .. path to xml-file containing the stack configuration (traceclasses etc.)
- NoScreenBoundary .. {0|1} if set, the controller can be moved outside the screen boundaries
- 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>…)
- 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

2.2.2.5 Start-List (Test environment)

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 2.2.2.4). Every line in the file contains the path to an application. You can use the prefix "min " to make an application starting minimized.

2.2.3 PCO-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 3.4 for detailed information on how to write your own PCO viewer.

The following paragraphs will give an detailed information about already existing and planned viewers.

2.2.3.1 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) – you just have to set the pre-processor define *CCD* to *0* or *1*. The first needs the ccddata_dll.dll to run which has to be build with the current cdg-files (if not delivered as well).

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

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 (This part is currently under development).

2.2.3.1.1 Classes and functions - under construction -

2.2.3.1.2 Sources - under construction -

The C/C++ sources of the standard PCO viewer can be found in the PCO directory in *src*, *src/view* and *src/view/std.* They are in detail:

• *IniFile.cpp*	
• *pco.ini*	default ini file (see 2.2.2.4)
• *pco2.bat*	batch file for starting default set of PCO components
• *pco_util.cpp*	common utility functions
• *syst_prim.cpp*	
• *view/TrcFmt.cpp*	
• *view/pco_view_core.cpp*	
• *view/pco_view_templ.cpp*	
• *view/std/dyn_array.cpp*	
• *view/std/win32/DlgOptions.cpp*	
• *view/std/win32/StdAfx.cpp*	
• *view/std/win32/dataReqDlg.cpp*	
• *view/std/win32/detailtreeview.cpp*	
• *view/std/win32/finddlg.cpp*	
• *view/std/win32/mfc_MainFrm.cpp*	
• *view/std/win32/mfc_doc.cpp*	
• *view/std/win32/mfc_view.cpp*	
• *view/std/win32/pco_view_std.cpp*	
• *view/std/win32/resource.h*	
• *view/std/win32/sendersdlg.cpp*	

- *view/std/win32/view.cpp*
- *view/std/win32/view.rc*
- *view/std/win32/xlistbox.cpp*

Include files used by the standard viewer are saved in *inc*, *inc/view* and *inc/view/std.* Their content is described below:

- *IniFile.h*
- *pco_const.h*
- *pco_util.h*
- *syst_prim.h*
- *view/TrcFmt.h*
- *view/pco_view_core.h*
- *view/pco_view_templ.h*
- *view/std/dyn_array.h*
- *view/std/win32/StdAfx.h*
- *view/std/win32/dataReqDlg.h*
- *view/std/win32/detailtreeview.h*
- *view/std/win32/dlgoptions.h*
- *view/std/win32/finddlg.h*
- *view/std/win32/mfc_MainFrm.h*
- *view/std/win32/mfc_doc.h*
- *view/std/win32/mfc_view.h*
- *view/std/win32/pco_view_std.h*
- *view/std/win32/sendersdlg.h*
- *view/std/win32/view.h*
- *view/std/win32/xlistbox.h*

The Win32-resources are situated in *res/view/std*:

- *Toolbar.bmp*
- *doc.ico*
- *view.ico*
- *view.rc2*

2.2.3.1.3 *Command line parameters*

On the command line of the standard PCO-viewer you may specify the following options which can be listed with "pco_view –h" as well:

-i <ini-file>	… use specified ini-file

2.2.3.1.4 *Ini-file settings*

Beside the general settings (see 2.1.4) 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)
- DataQueueSize … number of elements in queues for traces/primitives

2.2.3.2 Condat RT

The “Condat RT” project provides a whole database based test system for Layer2/3 traces and primitives. Via a port implemented as a PCO viewer (in detail derived from the core class) it can receive primitives from the PCO server.

2.2.3.3 MSCview

It is planned to enhance the “Message Chart Viewer” to be able to get primitives/traces directly from the PCO server. Currently it reads dbg-files generated by old PCO or the standard viewer.

2.2.3.4 Java Viewers

Since a native CMS-Java-Interface already exists it should be easy to implement any kind of JAVA viewer. The C++ base classes have not been adapted yet.

3 Interface specifications

The next paragraphs contain specific informations about used and provided interfaces.

3.1 CCD-edit - under construction –

If you build the standard viewer with *CCD=1* the ccdedit.lib (with functions for CCD decoding) and the *ccddata_dll.lib* (the import library *ccddata_dll.dll*, containing the specific data generated from the current cdg-files) will be linked to the executable. Furthermore code using the ccdedit interface will be compiled. The used functions are described below:

3.2 CMS

For detailed information about the Condat Multitasking System system and its interface see the manual: [CMS].

3.3 Controller-Server-View communication

As described before all communications between PCO components are implemented using the “Virtual Condat Multitasking System”. A control message is just a byte stream which can be interpreted using the class *Syst_prim* defined in *pco/inc/syst_prim.h* and *pco/src/syst_prim.cpp.* As you may guess by the name it is related to the system primitives used in the current Condat FRAME. In fact internal the struct *T_SYST_PRIM* defined in */GPF/INC/Header.h* is used. This is due to simplicity in the past because in former versions there was only one queue for control messages and traces/primitives from outside (testinterface). Maybe a new smaller header should be provided in future versions.

Anyway the important fields in the current header are:

- MsgId ... USHORT, the ID of the message (see PCO_XXX constants in *pco/inc/pco_const.h*)
- Data ... Byte stream, the content of the message which has to be interpreted due to the message type
- DataSize ... size of the data byte stream in bytes
- Sender() ... char[], name of the CMS-queue of the sender of this message

- Receiver( ) … char[], name of the CMS-queue of the receiver of this message

Usually the sender component creates a *Syst_prim*-object, gets a handle to the queue of the receiver (*q-open()*) and sends the object using the char*-cast operator (*q_write()*). The receiver component replies with PCO_OK (see 3.3.1.18) or PCOS_ERROR (see 3.3.1.8).

You'll find all defined (in *pco/inc/pco_const.h*) control message types with descriptions below:

3.3.1 Control messages

3.3.1.1 PCO_CLEAN

data:

purpose:
sent from viewer to server and then to all other viewers to inform them that the user has cleaned the viewer content

3.3.1.2 PCO_CLOSE_LOGFILE

data:

purpose:
sent to server to disable logfile-mode -> new state PCO_STOPPED.

3.3.1.3 PCO_CONNECT

data:

Char[] name of CMS-queue in which the viewer wants to receive traces/primitives (zero terminated)

purpose:
sent from viewer to server to establish connection, server adds client to its list

3.3.1.4 PCO_CONNECTED

data:

byte	server type id (see 3.3.3)

purpose:
sent from server to viewer to inform about an established connection and to tell its type

3.3.1.5 PCO_COPY_LOGFILE

data:

char[]	name of source sessionfile with full path (zero terminated)
char[]	name of destination sessionfile with full path (zero terminated)
LONG	first entry to be copied
LONG	last entry to be copied

purpose:
sent from viewer to server to make it copy a specified session into another sessionsfile while applying the filter currently set

3.3.1.6 PCO_DATA

data:

rawdata ... depends on server type

purpose:

sent from some servers to viewers (others send data without header)
contains rawdata which has to be interpreted depending on server type

3.3.1.7 PCO_DISCONNECT

data:

purpose:

sent from viewer to server to disconnect, server removes client from its list

3.3.1.8 PCO_DISTRIB_LOGFILE

data:

LONG	first entry to be sent
LONG	last entry to be sent

purpose:

sent to server (in logfile mode) to make him send logged data to all clients

3.3.1.9 PCO_ERROR

data:

U16	ID of message which has produced the error
byte	error code (see 3.3.2)

purpose:

sent by a receiver to the sender to inform about an error a received control message has raised

3.3.1.10 PCO_EXIT

data:

purpose:

sent to server to make it exit, server will send this message to all connected viewers before exiting

3.3.1.11 PCO_GET_LOGFILE_INFO

data:

Char[]	name of sessionfile with full path or just a session name (zero terminated)

purpose:

sent to server to request info's (e.g. count of entries) about the specified logfile

3.3.1.12 PCO_GET_LOGFILE_DATA

data:

ULONG	start index
ULONG	end index

purpose:

sent to server (which has to be in PCO_LOGFILE state) to make him forwarding all logged data from the current session file (which matches the index constraints) to the sender (has to be a connected viewer).

3.3.1.13 PCO_GET_SESSIONPATH

data:

purpose:

sent to server to get its current session path

3.3.1.14 PCO_GET_TESTSESSIONS

data:

purpose:

sent from controller to server to acquire a list of available session names (which are stored in current server testsession directory)

3.3.1.15 PCO_INIFILE_CHANGED

data:

purpose:

sent from viewer to server and then to all other viewers after an ini-file change which should be handled immediatly by all viewers

3.3.1.16 PCO_LOGFILE_COMPLETE

data:

purpose:

sent from server to a viewers after complete forwarding of a requested logfile

3.3.1.17 PCO_LOGFILE_INFO

data:

LONG	count of entries in logfile specified in last PCO_GET_LOGFILE_INFO
Char[]	optional name of ccddata-file used during logging (usually the .pco-file itself, zero term.)

purpose:

sent from server to a sender of PCO_GET_LOGFILE_INFO

3.3.1.18 PCO_OK

data:

U16	ID of message which will be confirmed by this PCO_OK

purpose:

sent by a receiver to the sender to confirm receiving and correct interpretation of a control message

3.3.1.19 PCO_OPEN_LOGFILE

data:

Char[]	name of sessionfile with full path or just a session name (zero terminated)
LONG	first entry to be sent
LONG	last entry to be sent

purpose:

sent from controller to server to open a file with logged data of a specified session -> new state of server: PCO_LOGFILE. (See also PCO_DISTRIB_LOGFILE)

sent from viewer to server to receive logged data of a specified session -> no change in state of server.

3.3.1.20 PCO_RENAME_LOGFILE

data:

char[]	original name of sessionfile (evtl. with full path, zero terminated)
char[]	new name of sessionfile (evtl. with full path, zero terminated)

purpose:

sent to server to make it rename a session logfile

3.3.1.21 PCO_SEND_PRIM

data:

char[]	receiver (zero terminated)
char[]	text (zero terminated)

purpose:

to request server to send a CONFIG-primitive to TST, not supported by all servers (knowledge of FRAME is necessary)

3.3.1.22 PCO_SESSIONPATH

data:

char[] list ... zero terminated path-string

purpose:

sent from server as reply to PCO_GET_SESSIONPATH

3.3.1.23 PCO_SET_FILTER

data:

char[] list ... zero separated entity names ("\0\0"==end)
e.g.: "MM\0RR\0SS\0\0" or "+"MM\0RR\0\0"
- first entity=="+" -> only this entities will be forwarded
- default: specified entities will not be forwarded

prim_trace ... optional parameter to disable/enable general forwarding of primitives/traces:
0 .. fowarding of everything (default)
1 .. no primitives
2 .. no traces

purpose:
sent from viewer to server to set the entity filter for this viewer

3.3.1.24 PCO_SET_SESSIONPATH

data:

Char[]	zero terminated path-string

purpose:
sent to server to set new session path

3.3.1.25 PCO_START_TESTSESSION

data:

Char[]	session name (zero terminated)

purpose:
sent to the server to start a new testsession -> new state PCO_RUNNING

3.3.1.26 PCO_STATUS

data:

<none>	.. if sent from controller to server

T_PCOSTATUS	.. current status of server (see T_PCOSTATUS)
Char[]	.. name of testsession (can be empty, zero terminated)

purpose:
sent from controller to server to request its current status,
sent from server to controller to publish its current status

3.3.1.27 PCO_SUBSCRIBE

data:

char[]	.. mobile ID (may be empty) (zero terminated)

purpose:
sent from viewer to server to receive live data of a dedicated mobile

3.3.1.28 PCO_STOP_TESTSESSION

data:

purpose:
sent to the server to stop a running testsession -> new state PCO_STOPPED

3.3.1.29 PCO_SYNCHRONIZE

data:

ULONG	time in ms

purpose:
sent from a viewer A to server to indicate change in the view to a new time stamp
sent from server to all viewers except A to synchronize them with A

3.3.1.30 PCO_TESTSESSIONS

data:

```
char[]     .. zero separated testsession names ("\0\0"==end)
USHORT     .. 1 - more messages will follow
              0 - last messages with testsessions
```

purpose:

reply to PCO_GET_TESTSESSIONS (see 3.3.1.14)

3.3.1.31 PCO_TO_FRONT

data:

purpose:

sent from GUI-controller to server to indicate activation of ctrl window
sent from server to all viewers to indicate activation of ctrl window
sent from server back to controller to indicate activation of viewers

3.3.1.32 PCO_UNSUBSCRIBE

data:

purpose:

sent from viewer to server to stop receiving of live data

3.3.2 Error codes

```
#define PCO_ERR_TSESSION_NOT_RUNNING 0

#define PCO_ERR_TSESSION_RUNNING     1

#define PCO_ERR_FILE_OPEN            2

#define PCO_ERR_SRV_RUNNING          3

#define PCO_ERR_FILE_NOT_OPEN        4

#define PCO_ERR_ALREADY_CONNECTED    5

#define PCO_ERR_NOT_CONNECTED        6
```

3.3.3 Server types

3.4 Deriving a new viewer

This chapter is intended to give potential developers of new PCO viewers a general description of how to do this.
Of course you can look into 3.3 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 "\GPF\INC" has to be added to the include path.
For the detailed description of the PCO_XXX control messages see 3.3.1 or "\GPF\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.
You may also review the sources of the standard command line viewer "pco_dump" (pco_dump.cpp, pco_view_cmdl.h and pco_view_cmdl.cpp).

3.4.1 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 "\GPF\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:

- "\GPF\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:

- "\GPF\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.

3.4.2 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 "\GPF\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 3.4.1). 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, U32 time, const char* sender):*

- 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
  time .. time stamp in ms since 1970
  sender .. name of sender

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:
- "\GPF\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:
- "\GPF\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.

3.4.3 Using PCOView_frameSupp

As stated in 3.4.2 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:

- "\GPF\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:

- "\GPF\LIB\WIN32\ccdedit.lib" ... contains functions for access to the "ccddata_dll.dll"
- "\GPF\LIB\WIN32\ccddata_load.lib" ... 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:

- "\GPF\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:

- "\GPF\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.

3.5 File formats supported by PCO

This chapter contains information about the different file formats which can be used together with PCO. After general introductions the details of the individual format will be explained.

3.5.1 *.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 **Error! Reference source not found.**) which has to be controlled to start/stop logging either by the command line PCO-Controller (see **Error! Reference source not found.**) or by the GUI-Controller (see **Error! Reference source not found.** 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 **Error! Reference source not found.** and [PCO_INTRO] chapter "Logging & Replay"), for filtered examination by loading into a Standard Viewer (see **Error! Reference source not found.**) 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 **Error! Reference source not found.**).
Parts of a logfile can be stored into new ones via the GUI-Controller (see **Error! Reference source not found.**) or the Standard Viewer (see **Error! Reference source not found.**).

Format details:
…

3.5.2 *.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 **Error! Reference source not found.**). 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.

Format details:
…

3.5.3 *.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 **Error! Reference source not found.** 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 2.2.2.5). You may specify a certain .svc-file as parameter to the Standard-Viewer on the command line (see also **Error! Reference source not found.**) or, e.g., drag and drop such a file to a running viewer to import the stored settings.

Format details:
…

3.5.4 *.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 **Error! Reference source not found.**) 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.

Format details:
…

3.5.5 *.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.

Format details:
…

3.5.6 *.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.

Format details:
…

4 Tests - under construction –

In this paragraph a list of possible software tests is presented which should/could be used to check and verify the functionality of the PCO components.

5 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.

5.1 Known bugs

•

5.2 „Soon implemented“

- GUI-interface to edit PCO controller start-list-file

5.3 „Nice to have“

- Online-Help
- Graphical selection of traceclasses and redirections
- Adaptation of the viewer core/template class to Java
- A new smaller header for control messages should be provided.

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/>