Bull DPX/20

Open Terminal Management (OTM) CPI-C SS Bull Environment User's Guide

AIX

ORDER REFERENCE
86 A2 32PE 04

Bull DPX/20

Open Terminal Management (OTM)
CPI-C SS Bull Environment User's Guide

AIX

Software

April 1996

Bull Electronics Angers S.A.
CEDOC
Atelier de Reprographie
331 Avenue Patton
49004 ANGERS CEDEX 01
FRANCE

ORDER REFERENCE
86 A2 32PE 04

Suggestions and criticisms concerning the form, content, and presentation of this book are invited. A form is provided at the end of this book for this purpose.

Trademarks and Acknowledgements

About this Book

This guide describes the Common Programming Interface for Communications Starter Set (CPI-C SS) which is an Application Program Interface (API) that is used with Bull's OTM (Open Terminal Management) product.

Who Should Use this Book

Application programmers use CPI-C SS to develop applications that run on Bull UNIX machines, and that work through Bull's OTM product to extract information from Bull mainframes running under the different GCOS operating systems.

The OTM Product

The OTM product covers the emulations necessary to connect Bull UNIX machines to other Bull machines that use Bull's GCOS proprietary operating systems as well as to IBM machines through the Bull/IBM gateway.

The OTM Manual Set

1. OTM Administrator and User's Guide, ref: 86 A2 31PE.
2. OTM TWS2107 Terminal Emulation User's Guide, ref: 86 A2 33PE.
3. OTM VIP7800 Terminal Emulation User's Guide, ref: 86 A2 34PE.
4. OTM CPI-C SS in Bull Environment User's Guide (emulation tailoring for applications), ref: 86 A2 32PE.
5. OTM and CPI-C SS Diagnostic Guide, Stack C, ref: 86 A7 52AJ.
6. The various Software Release Bulletins (SRB) delivered with each software release.

Software Requirements

OSI Stack layers.

The AIX Version 4.1 of UNIX.

The OTM product.

Organization of this Book

Chapter 1.	**Introduction** describes the CPI-C Starter Set.
Chapter 2.	**Using the CPI-C Starter Set** gives the procedures for using the CPI-C SS product with OTM.
Chapter 3.	**Variables, Functions and Verbs** describes these CPI-C Starter Set elements.
Appendix A	**Error Messages and Trace Facility** provides OTM error messages and return codes and their logging system.
Glossary	
Index	

Conventions

The generic term DPX is used throughout this guide, meaning by this DPX/20.

As OTM is available also on Bull DPX/2 systems, whenever the use of the generic term DPX could be misleading or not precise enough, the complete name is used (DPX/20 or DPX/2).

Prerequisite Publications

OSI Services Reference Manual (ref: 86 A2 05AQ)

Contents

Chapter 1. Introduction 1-1
Summary 1-1
User Visibility 1-1
What It Is and What It Does 1-1
What It Is 1-3
What It Does 1-4
Two Level Transaction Processing 1-4
Configurations Using the CPI-C Starter Set 1-5
Connection between DPX System and DPS 6000 1-5
Connection between DPX System and DPS 7000 1-5
Connection between DPX System and DPS 9000 System 1-6
Connection between Two DPX Systems 1-7
Connection between DPX System and IBM System 1-7
License Control – iFor/LS 1-8
Nodelocked License 1-8
License Control Prerequisites 1-8
CPI-C Starter Set License Control Implementation 1-8

Chapter 2. Using the CPI-C Starter Set 2-1
Summary 2-1
General 2-1
Tracing 2-1
Verbs 2-1
Mapped Type Conversation 2-1
Transparent Mode 2-2
Startup Procedure 2-3
Examples of User Application 2-5
First Example: DPX DPS x Connection 2-5
Second Example: DPX DPX Connection 2-7
Third Example: Incoming Connection from DPS 7000 2-8
State Transition Tables 2-10
Use of CPI-C Verbs in the COBOL MF Environment 2-12
Use of CPI-C Verbs in C Environment 2-13
Debugging 2-14
Generating a CPI-C SS Trace 2-14
Set Trace Level 2-14
Run CPI-C SS Application 2-15
Run scancpic Utility 2-15
Run dumpcpic 2-16
Trace Example 2-16

Chapter 3. Variables, Functions and Verbs 3-1
Summary 3-1
General 3-1
Variables 3-1
api_msg or rdmessg Function 3-3
retrieve_error Function 3-5
CMACCP 3-6
CMALLC 3-9
CMDEAL 3-11
CMINIT 3-13
CMRCV 3-16
CMSDT 3-20
CMSED 3-22
CMSEND 3-24
CMSERR 3-26

Appendix A. Error Messages, Return Codes and Trace A-1
Summary A-1
Error Messages A-1
  Legend A-1
  CPI-C Starter Set Configurator Messages A-1
Return Codes A-4
CPI-C SS Trace Facility A-5
  Trace Levels A-5
  Trace Settings A-6

Glossary G-1

Index X-1

Chapter 1. Introduction

Summary

– User Visibility, on page 1-1.

– What It Is and What It Does, on page 1-1.

– Two Level Transaction Processing, on page 1-4.

– Configurations Using the CPI-C Starter Set, on page 1-5.

– License Control – iFor/LS, on page 1-8.

User Visibility

CPI-C SS (Common Programming Interface - for Communications Starter Set) creates applications to run through the Bull Open Terminal Management (OTM) product to extract information from Bull and IBM mainframes. CPI-C SS configuration requires only a few additional operations after OTM configuration. Terminal management is completely performed by OTM. The remote system continues to see the other end of the connection as a terminal. The current level of security of the host system is not modified.

On the local system, it is only necessary to define the characteristics of the conversations and to administer the connections through a menu driven procedure.

CPI-C SS coexists with the functionality offered by the Terminal Manager. The terminal operator and application on the local system can establish concurrent sessions with the host.

CPI-C SS sees data in transparent mode; the user application on the local system is totally responsible for the handling of incoming data from host applications which see the correspondent application as a terminal. The data display is determined by the terminal type defined in the configuration.

The data length is the value of the SSDU (Session Service Data Unit) supported by OTM. The maximum value is 18 KB. The SSDU value depends on the application running on the remote host. The header and terminator characters for the VIP protocol are not visible.

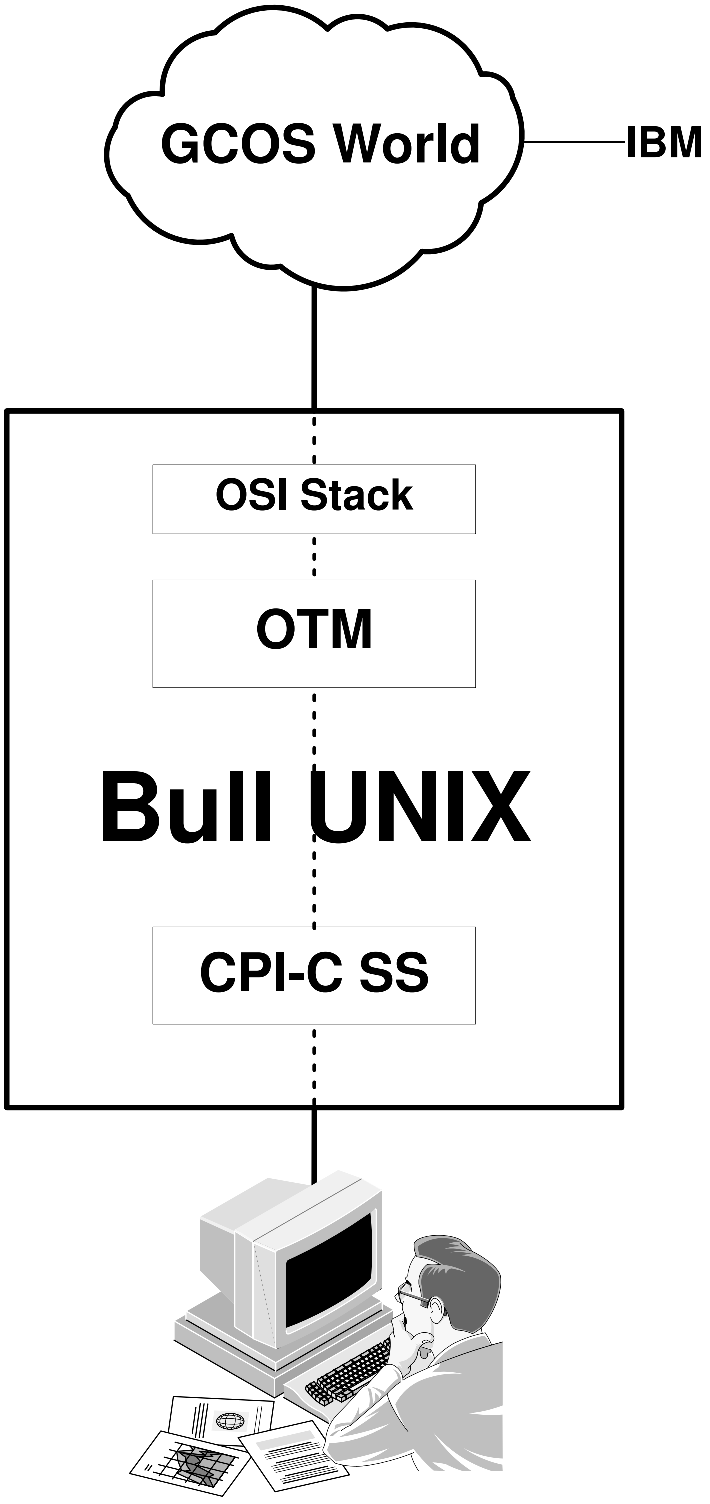
Security of file updating in case of abnormal termination is left to the application programmer.

The CPI-C library maintains one conversation per application.

There is no limitation on the number of conversations that the library can maintain at the same time. Limitations on this number may be imposed by limitations on lower layers (eg. maximum number of ISO sessions).

What It Is and What It Does

CPI-C SS is an Application Program Interface (API) used by application programmers on Bull UNIX machines to write programs to extract information from Bull mainframes running under the GCOS operating systems and also from IBM mainframes through the Bull/IBM gateway.



CPI-C Starter Set Develops Applications to Communicate with Bull GCOS Machines

What It Is

The CPI-C interface is defined by the X/OPEN UNIX standards setting organization.

In the CPI-C syntax, each API function is called a ”VERB”. Each verb has a function name starting with ”CM” – for example ”CMSEND” and ”CMRCV”. There are between 30 and 40 verbs in the complete X/OPEN CPI-C definition. However X/OPEN defines 6 of these verbs as being ”basic”. This group of 6 verbs is called the ”Starter Set”.

The Starter Set functions defined by X/OPEN are:

Mnemonic	Verb Name
Initialize_Conversation	CMINIT
Allocate	CMALLC
Accept_Conversation	CMACCP
Send_Data	CMSEND
Receive	CMRCV
Deallocate	CMDEAL

The CPI-C SS product implements all the above verbs, which is why it is called ”CPI-C Starter Set”. The syntax and the parameters of each verb are as defined by X/OPEN. However not all the options of each verb may be supported. For example it is not possible to perform two consecutive calls to CMSEND, since CMSEND performs the same function as ’TRANSMIT’ on a synchronous terminal and will cause the ”Token” to be given to the remote application.

The CPI-C SS product also implements the following verbs, which are not part of the ”Starter Set” of CPI-C verbs, but which are defined by X/OPEN:

Mnemonic	Verb Name
Set_Deallocate_Type	CMSDT
Set_Error_Direction	CMSED
Send_Error	CMSERR

The CPI-C SS product also includes two diagnostic functions, not defined by X/OPEN and thus not compatible with other CPI-C interfaces. These functions are:

```
api_msg(int keymesg, char *datafile)
```

This function generates a text message describing the return code generated by a CPI-C SS function call. Thus when called with the values:

```
api_msg(1, ””);
```

it returns a pointer to the following text message:

```
”CM_ALLOCATE_FAILURE_NO_RETRY”
```

```
retrieve_error()
```

This function retrieves detailed diagnostic information from the service provider (OTM) and uses this information to generate a text message. It takes no parameters.

An example of a message that might be generated by this function is:

```
”4644   2ltp load_conv: Entry TOTO not found in configuration file
             /usr/CPI-C/site.cnf”
```

What It Does

Using CPI-C SS, an application programmer creates simple applications for extracting information from the mainframes.

CPI-C SS creates programs using two level transaction processing. The application at one end of the link (subordinate) is seen at the other end, by the TP (Transaction Processing) system, as a normally supported terminal, sending and receiving messages.

The application running on the DPX system (subordinate) gains access to applications running on GCOS 6, GCOS 7, GCOS 8, and IBM systems.

In order to support these connections, OTM must be installed, to provide terminal management functions.

CPI-C SS is very closely linked to OTM. Thus many people using OTM will need to use CPI-C SS at one time or another.

The ”service provider” used by CPI-C SS is OTM. OTM uses the OSI stack to communicate with BULL DPS Mainframes and Minicomputers, IBM Mainframes, and with BULL UNIX Computers. OTM uses the services offered by the Session layer in the OSI stack.

Since the service provider used by CPI-C SS is OTM, the services offered to the application program are ”Terminal–orientated”. The remote application on the remote machine is never able to know whether it is dialoguing with a real physical terminal, or with an application program.

Two Level Transaction Processing

CPI-C SS applications function using ”Two Level Transaction Processing”. Sometimes you may see 2LTP or 2L–TP. 2LTP was at one time the name of the CPI-C SS product. This was due to the fact that the CPI-C SS application is always seen as a terminal. In a host–terminal situation, the two ends of the connections are not considered ”equal”. The application running on the host is considered to be the ”Master” of the dialogue, while the terminal is considered the slave. In addition the Host application has some ”rights” that the terminal does not have, for example the use of the token, requests to close the connection, etc..

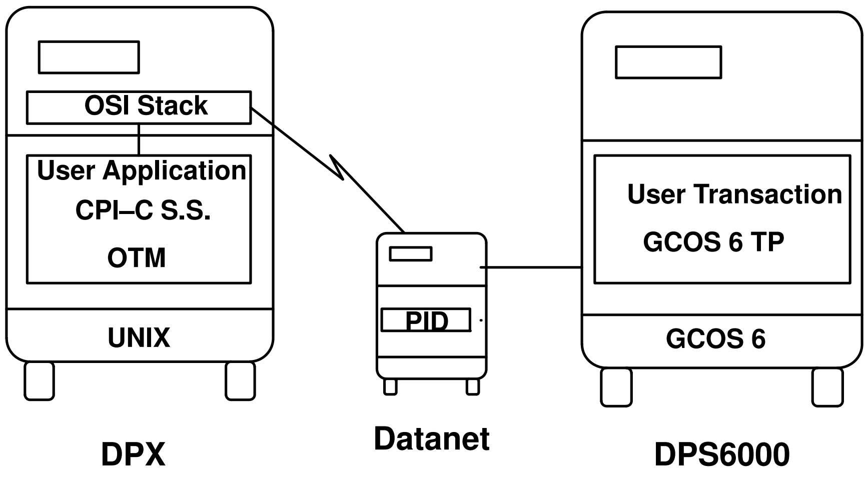
Thus when a CPI-C SS application is communicating with an application on a host, it is said that the application is ”Two Level”, because one application does not have as many ”rights” as the other.

Configurations Using the CPI-C Starter Set

Examples of possible configurations follow.

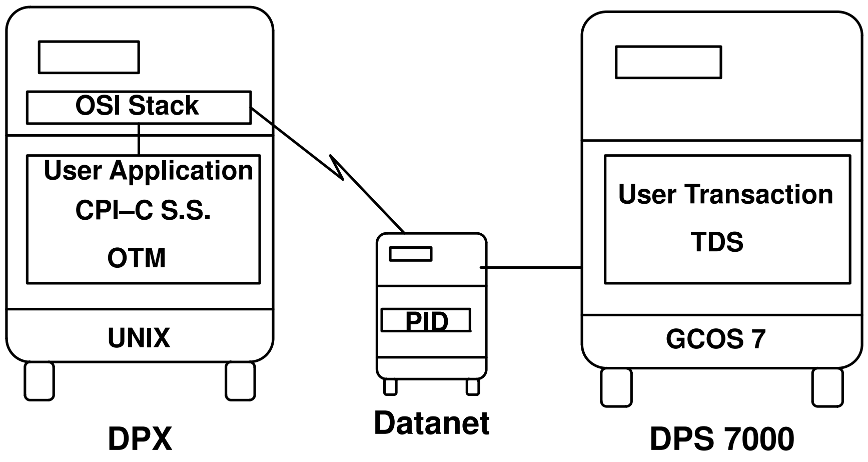
Connection between DPX System and DPS 6000

The DPX application is always the initiator of the conversation with the application running under the GCOS 6 TP (for example, DMVITP or TPS6 or DTF or ECL) control.



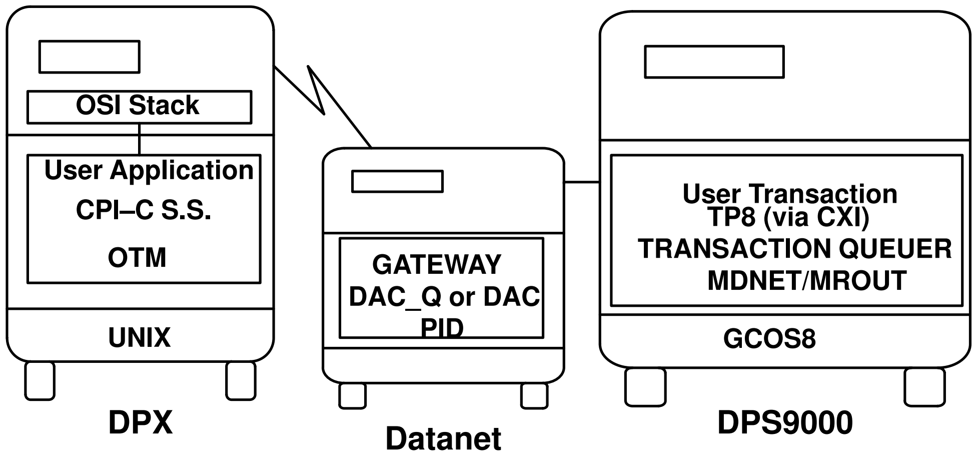
Connection between DPX System and DPS 7000

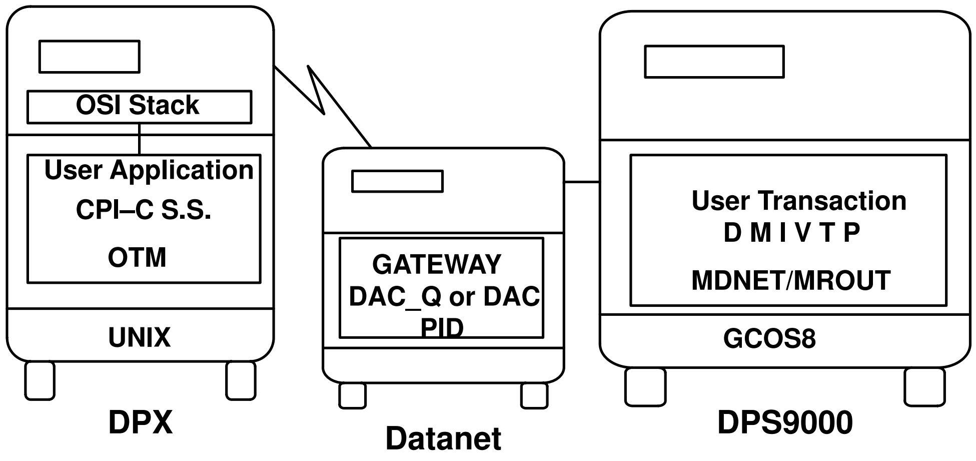
The DPX application is always the initiator of the conversation with the application running under the GCOS 7 TP (for example TDS or IOF) control.



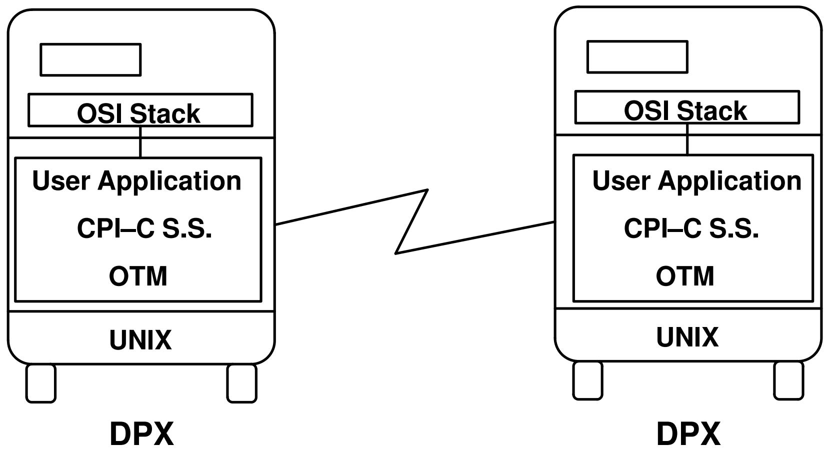
Connection between DPX System and DPS 9000 System

The DPX application is always the initiator of the conversation with the application running under TP8 (via CXI) or TS8 or TSS or DMIVTP control.



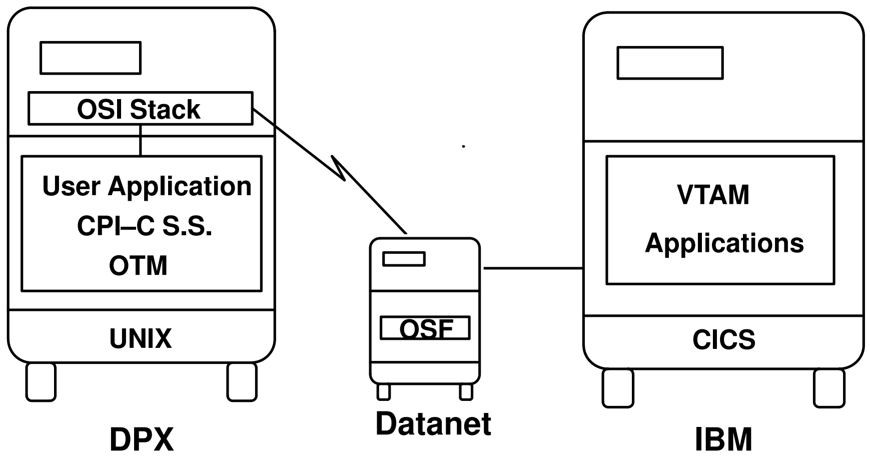


Connection between Two DPX Systems



The connection between two DPX systems may also take place via a DataNet.

Connection between DPX System and IBM System



License Control – iFor/LS

The Network Licensing System uses encrypted licenses, to manage software products to maintain compliance with terms of licensing agreements.

Nodelocked License

The Licensing model used with CPI-C Starter Set is ”nodelocked”.

Nodelocking (also known as CPU locking) is a licensing mechanism requiring each node (workstation) on which the licensed software operates to obtain an authorised license for its unique System ID.

License Control Prerequisites

The prerequisites which apply to the licenses are derived from functional prerequisites.

To use CPI-C Starter Set it is mandatory to have previously installed OTM and OSI Stack upper and lower layers (osi_frame, osi_low, osi_high) and therefore the associated licenses.

CPI-C Starter Set License Control Implementation

CPI-C Starter Set needs a license in nodelocked mode to work.

When CPI-C Starter Set license is absent or expired, the connections to OTM through CPI-C are refused. For incoming connections, a CM_ALLOCATE_FAILURE_NO_RETRY is returned by the CMACCP verb.

In addition, the ”4697 2ltp Accept_C: CPI-C Starter Set license not available.” is written in the ”/tmp/api.trc” file.

For outgoing connections, a CM_ALLOCATE_FAILURE_NO_RETRY is returned by the CMINIT verb.

In addition, the ”4698 2ltp Initiate_C: CPI-C Starter Set license not available.” is written in the ”/tmp/api.trc” file.

The current connections are not affected by the expiration of the CPI-C Starter Set license.

The absence of OTM license will lock the OTM daemon and so all TM-USERS new connections will be rejected.

The absence of the OSI layers licenses will lock the top and bottom sub–components of the OSI layers: session, transport, LLC and X25.3 mapper.

Chapter 2. Using the CPI-C Starter Set

Summary

– Debugging, on page 2-14.

General

The CPI-C SS programming interface is based on a subset of X/Open CPI-C/OSI primitives set.

It is available in the **libotmapi.a** library, under the **/usr/lib** directory.

Under the /**usr**/**include** directory, there are two header files: **CPI-C.h** for the C language, and **CPI–H** for the Cobol language.

Applications written using these libraries, are able to maintain a conversation with user applications running on the remote system.

Tracing

These libraries provide tracing capabilities to help the user for application debugging. A quick test procedure, using library verbs one by one, is provided to help the application programmer test the connections. See Chapter 3, ”Quick Test Procedure” in the *OTM Administrator's and User's Guide*.

Verbs

COBOL 85 (MICROFOCUS COBOL II) and C programs can invoke the CPI-C verbs. The verb syntax, parameters, functionning and return codes are in accordance with X/Open specifications.

The implemented CPI-C verbs are:

CMINIT – initialize conversation

CMALLC – allocate

CMACCP – accept conversation

CMSEND – send data

CMSED – set error direction

CMRCV – receive information

CMSERR – send error

CMSDT – set deallocate type

CMDEAL – deallocate

Mapped Type Conversation

The mapped conversation behavior is used. In this way programs can exchange data records with a data format defined by the application programmers; data is transmitted in a transparent way, and the presentation rules of the data stream are honored in relation to the terminal type.

Transparent Mode

The *Transparent Mode* is used to decompose a function into its elemental parts (bypassing the CPI-C rules).

The *Transparent Mode* is enabled by setting the environment variable

```
API_MODE = TRANSPARENT.
```

The CMALLC can be decomposed into the following steps:

- *connect request*
- *connect confirm*

The connection state can be Receive or Send depending on the application being called.

During a debugging phase, *Transparent Mode* is used to analyze the error that occurred.

Startup Procedure

The login procedure can be modified by the user.

A file containing login data, which are to be sent to the remote system, is provided to perform the login procedure in NON TRANSPARENT mode. The file name is specified in the environment variable AUT_LOG_FILE. If the variable does not exist the automatic login procedure is performed.

This file is created with the **cpic_startup** command. The user is prompted to enter the values of the different parameters. The user then enters the name of the created file in the AUT_LOG_FILE variable with this command:

```
export AUT_LOG_FILE=$HOME/file_name
```

The file contains two commands:

1. Send Data
2. Receive Data (and check if send has been completed successfully)

It has the following structure:

```
R%*
S%1<user name>
R%*
S%2<user password)
R%*
S%3<project>
R%*
S%4<billing>
R%*
```

More than one Send Data command can be specified in this file.

A Send Data command must always be followed by a Receive Data command.

The Send Data command has the following format:

```
<cmd>%1<data>%2<data>%3<data>%4<data>
```

<cmd> is the command name Send Data (S)

%1–%4 identify the arguments of the command. A maximum of 4 arguments can be specified. If present each identifier contains the following:

%1 = user name

%2 = user password

%3 = project

%4 = billing

These arguments are in the file: `site.cnf`

<data> is a sequence of characters to be sent to the remote system to perform the login procedure.

Special characters can be specified. For example \c \ n \ t \ v \ b.

The Receive Data command checks if the Send Data command has been completed successfully, that is, if the login procedure has been performed and the prompt string has been returned.

The format is the following:

```
<cmd><data>
```

<cmd> is the command name Receive Data (R)

<data> is a sequence of characters, usually the system prompt, that the system has to display after the login procedure has been performed. This string of characters is used to check the answer from the remote system. If the answer contains the specified string the S command has been completed successfully. If the answer does not contain the specified string the S command failed: CM_ALLOCATE_FAILURE_NO_RETRY will be returned to the application.

If <data> = %* no check is performed on data that are to be received.

Special characters can be specified, for example \c, \n, \t, \b, \v.

After the execution of the commands contained in the file the user application must be in SEND state. This condition occurs if the last executed command is a Receive command. If the last executed command is not a Receive command, this message is returned to the application: CM_ALLOCATE_FAILURE_NO_RETRY.

Note: The API_MODE environment variable must be undefined in order to use NON TRANSPARENT mode, which is the default mode in CPI-C SS.

Examples of User Application

To exchange data (to synchronize) between an application running on DPX and an application running on GCOS x or DPX host or IBM host, the dialog must be performed as explained in the following examples.

Data exchanging means to send commands to be executed to the correspondent and to receive an answer from the correspondent about the executed commands.

First Example: DPX DPS x Connection

In this example the CPI-C Starter Set program plays the role of a terminal (screen) application.

The behavior of the remote system GCOS x, when the CPI-C commands in the local application are executed, is explained in the following table:

Local Application	GCOS 8 Behavior
To initialize the conversation characteristics, CMINIT must be called into the local application. A conversation–ID is returned by CMINIT.	
To establish the conversation connection, CMALLC must be invoked with the conversation–ID assigned by CMINIT.	
	After executing the CMALLC, a Connect Indication is received by the GCOS 8 system. A Connect Confirm must be returned to the local application.
	For GCOS 8, data (first selection form) is sent to the local application and the turn is given to the local application.
	For GCOS 8, data (login key request) must be sent to the local application and the turn is given. The local application returns login parameter (user$password) to the GCOS 8 and turn is given to the GCOS 8. GCOS 8 sends the first selection form to the local application and turn is given to the local application.
After executing the CMALLC, the conversation is established, the local application is in send state. A CMSEND, used to send the request of TP (typically a command), must be executed. The local application is in receive state.	
	Data (the command) is received by the GCOS 8 system and turn is given to GCOS 8. The system executes the command and it sends the result to the local application. The TP form (the result) is returned to the local application.

After the CMSEND the state is receive.	
The local application must invoke a CMRCV to wait for incoming event (the result) that is already present. The local application is in send state.	
Many CMRCVs are necessary as the data length to be received from GCOS 8.	
	GCOS 8 returns to the local application the result of the executed command and after the result GCOS 8 sends the turn to the local application.
A CMSEND must be invoked to send TP input data	
	Data are received by the system and turn is given to the local application. The form is returned to the local application.
The local application must invoke a CMRCV to wait for incoming event that is already present. The local application is in send state.	
A CMSEND must be invoked to send logoff key.	
A CMRCV must be called to wait for incoming event.	
	Data are received by the system and turn is given. A session release request is sent to the local application. A session release confirm is received by the system.
The session release request is received by the local application and a return code CM_DEALLOCATED_NORMAL means that the conversation is deallocated.	

The programming schema between the DPX applications and the GCOS x remote system can be the following:

Local Application on DPX	**GCOS 8 Behavior**
.	
CMINIT	
.	
CMALLC	GCOS 8 receive connect indication send connect confirm send login key + turn receive user$password + turn send selection form + turn
.	
CMSEND	receive data + turn
CMRCV	send the TP form
	.
CMRCV	send the TP form + turn
CMSEND	receive data + turn

CMRCV	send the form
CMSEND logoff key	receive data + turn
CMRCV	send session release request
rc = CM_DEALLOCATED_NORMAL	receive session release confirm
.	exit

The host application must accept a login string in the format **user $ password**. Otherwise the login sequence must be provided by the 2LTP application (using a CMSEND).

Second Example: DPX DPX Connection

In this example the CPI-C Starter Set program plays the role of a terminal (screen) application.

The behaviour of the remote system DPX, when the CPI-C commands in the local application are executed, is explained in the following table. The environment variable **send type** is set to SEND AND PREP TO RECEIVE.

Local Application	Remote Application
To initialize the conversation characteristics, CMINIT must be called by the local application. A conversation–ID is returned by CMINIT.	
	To accept an incoming conversation request CMACCP must be called by the remote application, that is waiting for remote conversation requests.
To establish the conversation connection, CMALLC must be invoked with the conversation ID assigned by CMINIT.	
	After CMACCP has been completed successfully a conversation-ID is returned to the remote application.
After executing the CMALLC, the conversation is established, the local application is in SEND state. The remote application is in RECEIVE state.	
A CMSEND, used to send data to correspond, must be executed	
	The remote application must invoke a CMRCV to wait for incoming event that is already present. The local application is in Receive state and the remote application is in Send state.
	A CMSEND must be invoked to send data to correspond.
The local application must invoke a CMRCV to wait for incoming event that is already present. The local application is in Send state and the remote application is in Receive state.	
	A CMRCV must be called to wait for incoming event.
A CMSEND must be invoked to send data.	

	Data are received by the remote application that is in Send state. The local application is in Receive state.
The local application executes a CMRCV to wait for incoming event.	
	The remote application executes a CMDEAL to communicate the end of the conversation.
The local application receives a return code CM_DEALLOCATED_NORMAL meaning that the conversation is deallocated.	

The programming schema between two CPI-C applications running on two DPX systems can be the following:

Local Application on DPX	**Remote Application on DPX**
.	
.	
.	
CMINIT	
	CMACCP
CMALLC	
CMSEND	
	CMRCV
	CMSEND
CMRCV	
	CMRCV
CMSEND	
CMRCV	
	CMDEAL
rc = CM_DEALLOCATED_NORMAL	

Third Example: Incoming Connection from DPS 7000

In this example the CPI-C Starter Set program plays the role of a terminal (printer) application.

Perform the following:

1. Create the remote site to be accessed using smit.

   For example:

   ```
   Site Name               remsite
   Remote DSA Site         PH79
   ```

2. Create the session user using smit.

   For example:

   ```
   Remote Session User     dst8
   Site Name               remsite
   Mailbox                 DST8
   ```

3. Create the Symbolic Destination name using smit.

   For example:

   ```
   Symbolic Destination name       PRDST8
   Terminal Type                   VIP7800N
   Remote Session User             dst8
   ```

4. Set the API_TERM_NAME variable as follows:

   ```
   API_TERM_NAME=PRDST8; export API_TERM_NAME
   ```

5. Create a CPI-C Starter Set program as follows:

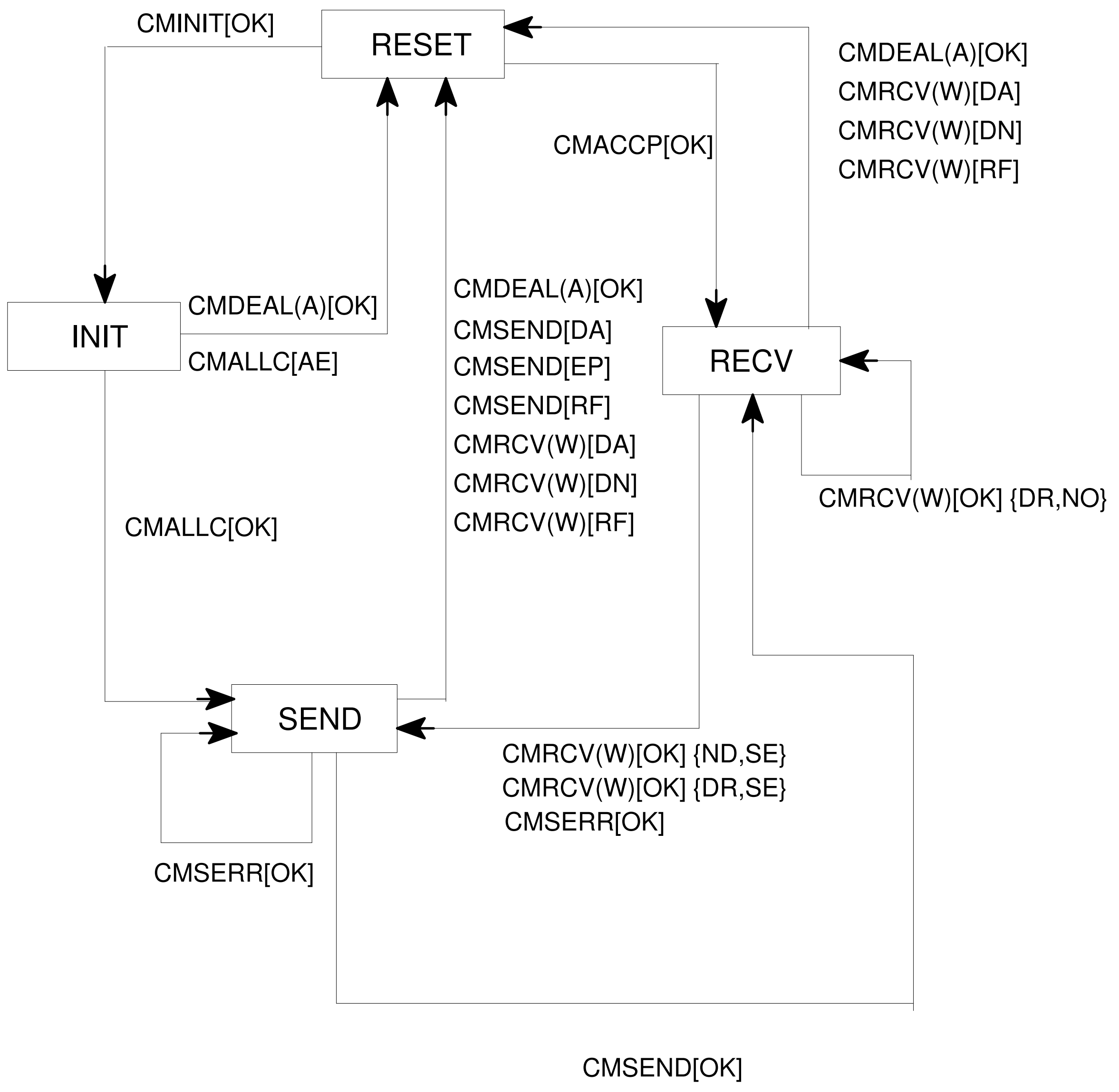
CMACCP	the program is waiting for data from DPS 7000
CMRCV	the program is in receive state. (Note that the state of the program depends on the host sending data). The program receive data and when CMRCV finished receiving data it exits with STATUS RECEIVED
CMSEND	the CPI-C Starter Set program sends the token; This confirms to the DPS 7000 that the program received data and then closes the connection

6. Launch the CPI-C Starter Set program.
7. To verify that OTM opened a connection (DST8), launch inftms. The connection is waiting for incoming data.
8. Connect to the DPS 7000 host (using a CPI-C Starter Set program or a tmcall) and create a report with destination **DST8** (configured on the DPS 7000 as remote printer).
9. The report is managed by the CPI-C program: It can be stored on file or it can be sent to the printer or filtered by an user program.

State Transition Tables

During the execution phase, the local application changes its state each time a CPI-C call is issued. Each CPI-C call causes the local application to change from a certain state A to another state B. If the local application is not in the proper state when the CPI-C routine is invoked, a return code indicates the error that occurred. If an error is detected, the state transition does not occur.

For each state, the figure illustrates the behaviour of each CPI-C function.



Note that initial state is ”RESET”

Error ”CM_PROGRAM_PARAMETER_CHECK” does not change state in any case.

Error ”CM_PROGRAM_STATE_CHECK” is returned if action cannot be performed in current state.

The symbols enclosed in round brackets indicate a conversation characteristic. The symbols enclosed in square brackets indicate a return code. The symbols enclosed in curly brackets indicate the value of the parameters data received and status received. The symbols have the following meaning:

A	deallocate_type = CM_DEALLOCATE_ABEND
W	receive_type = CM_RECEIVE_AND_WAIT RS
RS	receive_type = CM_REQUEST_TO_SEND_RECEIVED AE
AE	CM_ALLOCATE_FAILURE_NO_RETRY DA
DA	CM_DEALLOCATE_ABEND DN
DN	CM_DEALLOCATE_NORMAL EN
EN	CM_PROGRAM_ERROR_NO_TRUNC EP
EP	CM_PROGRAM_ERROR_PURGING PE
PE	CM_PARAMETER_ERROR PC
PC	CM_PARAMETER_CHECK RF
RF	CM_RESOURCE_FAILURE_NO_RETRY DR
DR	CM_COMPLETE_DATA_RECEIVED ND
ND	CM_NO_DATA_RECEIVED NO
NO	CM_NO_STATUS_RECEIVED SE
SE	CM_SEND_RECEIVED

Suppose that the local application is in Send state (SEND). A CMRCV is issued and receive_type is set to CM_RECEIVE_AND_WAIT. After the CMRCV execution the following situations can occur:

- if a return_code = CM_DEALLOCATE_ABEND
  - or a return_code = CM_DEALLOCATE_NORMAL
  - or a return_code = CM_RESOURCE_FAILURE_NO_RETRY
  - is returned the local application enters Reset state (RESET).
- if a return_code = CM_PARAMETER_CHECK is returned the local application does not change state

Suppose that the local application is in Initialize state (INIT.). A CMDEAL is issued. The following situations can occur:

- If deallocate_type is set to CM_DEALLOCATE_ABEND, after the CMDEAL execution the local application enters Reset state and a return_code = OK is returned.
- If deallocate_type is set to CM_DEALLOCATE_ABEND, after the CMDEAL execution the local application does not change state and a return_code = CM_PARAMETER_CHECK is returned.

Use of CPI-C Verbs in the COBOL MF Environment

Micro Focus COBOL is composed of four principal topics: COMPILER, NATIVE CODE GENERATOR, RUN TIME SYSTEM (RTS), LINKER. CPI-C verbs that are used in user applications, are in the libotmapi.a library. In this environment, two executions mode for cobol programs are allowed; they can be statically linked (applications) or dynamically called.

These verbs can be used both for statically linked programs and dynamically called programs. In both cases the execution environment must be created.

For statically linked programs, it is sufficient to link RTS, CPI-C library and the application program to obtain the object. For example:

```
cob –x prgname.cbl –o exname –L/pathname –llibname
```

This command produces an ”Object File Format” (OFF), that can be executed entering:

```
exname
```

For dynamically called programs, it is necessary to link to the RTS environment the part of the CPI-C library involved with the application program, to obtain the object. To correctly link RTS and the CPI-C library functions involved with the program, CCPSTID.o object is provided.

For example:

```
cob –xe ”” /usr/CPI–C/CCPSTID.o –L /usr/lib \
–lotmapi –o  rts32
```

where **rts32** is the name of the standard RTS provided by MicroFocus.

After this command has been executed, an extended RTS that includes CPI-C verbs is generated.

rts32 must be under the directory specified by the environment variable COBDIR and replaces the old **rts32** .

The command *cobrun* executes the user application program, (See the *COBOL/2 MF Operating Guide*), and calls *rts32* .

Programs are executed under RTS control. RTS can dynamically call the *.int* files (Intermediate Code File generated by the COMPILER) or the *.gnt* files (Generate Native Code produced by NATIVE CODE GENERATOR).

To execute subprograms written in non–COBOL languages (e.g. C) with the COBOL verb *call* , it is necessary to link such subprograms to the RTS or to the COBOL application.

The command that generates an extended RTS or an application, is:

```
cob [options] filename ...
```

where *filename* is one or more files having the extension:

```
.cbl (or .CBL or .cob),  .int  .gnt  .c  .o  .a  .s
```

The execution of the *cob* command produces an *end–point* which may be:

```
.gnt    ”dynamic load” mode execution (–u option)
.int    ”dynamic load” mode execution (–i option)
```

a statically linked module, type OFF (-**x** option)

(See *COBOL/2 MF Operating Guide* Chapter 9).

The *endpoint* **.gnt** that uses the CPI-C verbs, is executed under the new RTS.

The extended Run Time System can be assigned an other name chosen by the user. For example:

```
cob -xe "" /usr/CPI-C/CCPSTID.o -L /usr/lib \
-lotmapi  -o  rts32
```

The user application program that uses CPI-C verbs is *myprog.cbl*. To compile *myprog.cbl* and to obtain a **.gnt** file, use the command:

```
cob  -u myprog.cbl
```

To execute the user application program:

```
myrts myprog
```

Use of CPI-C Verbs in C Environment

CPI-C verbs that are used in user applications are in the *libotmapi.a* library. To generate an object file perform the following:

```
cc -O -I/usr/mydir cpicpgm.c -o cpic  -bI:/usr/lib/libotmapi.exp
```

Where:

I/usr/mydir is a private include directory.

bI:/usr/lib/libotmapi.exp is the export file for the shared library **/usr/lib/libotmapi.a** .

No other OTM library must be included because libotmapi.a automatically calls **libtmpi.a** and **libmef.a**.

Debugging

Since CPI-C SS uses OTM, debugging a CPI-C SS application often means debugging an OTM problem.

CPI-C SS is simple to debug. There is a function that sends a message from OTM which describes exactly ”why” the last error occurred. Also, there is an ASCII file (/tmp/api.trc) that contains similar information.

If you need more detailed information on the CPI-C SS Trace Facility, refer to Appendix A.

Generating a CPI-C SS Trace

In addition a very easy-to-use (or more accurately ”easy to understand”) trace facility exists.This trace system is particularly useful to application developers, because it shows all the parameters and data exchanged between the API and the application.

The CPI-C SS traces are designed to be very similar to the ”CPI-C OSI user traces”. However the method of activating and stopping them is different.

The simplest way to generate a trace is to proceed as follows:

1. Set up the trace level for the ”symbolic destination name” using SMIT.
2. Run the CPI-C SS application.
3. Run the utility ’scancpic’.
4. Run the utility ’dumpcpic’.

See Appendix A, *Error Messages and Trace Facilit*y, for further information on the CPI-C SS Trace Facility.

Set Trace Level

Run ’smit OTM Configuration menu’ and make the following choices:

CPI-C Starter Set Configuration

Change Symbolic Destination Description

Use ’F4’ to display all the Symbolic Destination descriptions. Choose the one that is used by the CPI-C SS application to extract the configuration information.

When the ’Change Symbolic Destination Description’ menu is displayed, modify the fields ”Trace Level” and ”Trace Storage Mode”. Set the trace level to ’2’ or ’4’ (0 and 1 will produce no trace, 3 produces the same trace as 2). Set the trace storage mode flag to 1.

Please see the OTM Administator and User’s Guide and the OTM Diagnostic Guide for more detailed information about the Trace Level and Trace Storage Mode.

When the new values have been entered, hit <Enter> to validate the new configuration.

The procedure described above allows the configuration file to be updated. However the new configuration information is not immediately usable. By default it is only activated the next time the machine is rebooted.

The following sequence of commands allows the new CPI-C SS configuration to be used by the next CPI-C SS application that is executed:

Run the SMIT ’OTM Configuration menu’ and make the following choices:

CPI-C Starter Set Configuration

Load New Symbolic Destinations Configuration

The following question is displayed:

Configuration Updating – Are You Sure ?   [y]

Leave the default answer and validate with <Enter>.

The trace levels are now set up correctly.

Run CPI-C SS Application

Execute the CPI-C SS application normally. No special configuration is required. The trace information generated is written in a binary format to the file '/tmp/apilog'.

Run scancpic Utility

The utility 'scancpic' generates an ASCII file identifying all the 'Conversation IDs' that are in the trace file ('/tmp/apilog').

The purpose of this step is to allow the trace information corresponding to one specific connection to be extracted.

The following example shows how 'scancpic' is used:

```
$ scancpic -?
usage: scancpic -i <in_file> -o <out_file>

$ scancpic -i /tmp/apilog -o /tmp/scancpic

$ cat /tmp/scancpic
****************************************************************
* Do not remove the comments from this file. The only thing you *
* can do with this command is to delete the lines in which are  *
* desribed pseudo conversation IDs that do not concern you.     *
* Before running dumpcpic plese remove the invalid entries      *
* (Invalid ID because trace level is less than 2).              *
* PSEUDO CONVERSATION IDS            START OFFSET IN INPUT FILE *
****************************************************************
          58e70000                                  00000000
```

There is no real need to modify the contents of the output file, unless the trace file '/tmp/apilog' is very large and there are a lot of "Pseudo Conversation IDs" listed.

Run dumpcpic

The 'dumpcpic' utility uses the ASCII file produced by 'scancpic' and the binary CPI-C SS trace file ('/tmp/apilog' by default) and produces an ASCII output file ('/tmp/dumpcpic' by default). Here is an example of how it can be used:

```
$ dumpcpic -?
usage: dumpcpic [ -i <inpput_file> ] [ -p <path-file> ] { -c
<conv_id> } [ -f <scanfile> ] [-s] [-e]

$ dumpcpic -f /tmp/scancpic
Dumping the file /tmp/dumpcpic
  ....
End of dumping the file /tmp/dumpcpic

$ head /tmp/dumpcpic
TRACE CPIC      file:  dumpcpic    date: Wed Feb  2 19:21:05 1994
                level: DACTRC= 4
                path:  DACTRD= /tmp

+----------------------------------------------------------------+
|TRACE CPIC verbs with input parameters at beginning of each    |
|verb entry then retuned values at the end of verb processing   |
+----------------------------------------------------------------+


CMACCP
        Input Conversation ID = 00000000
        ACCEPT: Conversation Status == S_RESET
        ACCEPT: calling function utm_acpt().
        ACCEPT: calling function get_msg().
```

Use InfoExplorer to search the Documentation CDROM for more information on these commands.

Trace Example

Here is an example of the traces that might be generated by a call to the 'CMINIT()' function.

```
TRACE CPIC    file:  dumpcpic      date: Mon Jan 31 16:19:59 1994
              level: DACTRC= 4
              path:  DACTRD= /tmp

+----------------------------------------------------------------+
|TRACE  CPIC verbs with input parameters at beginning of each   |
|  verb entry then retuned values at the end of verb processing |
+----------------------------------------------------------------+

CMINIT
        Input Simbolic Destination Name = LOOPPRT1
        Input Conversation ID = 00000000
        INITIATE: State == S_RESET
        LOAD_CONV: Cofiguration found LOOPPRT1
        LOAD_CONV: Term=DKU7107
        LOAD_CONV: Log tmpi_lev=-1
        LOAD_CONV: Remote Session=PRT1
        LOAD_CONV: User=USER
        LOAD_CONV: Password=
        LOAD_CONV: Billing=
        LOAD_CONV: Project=
        LOAD_CONV: Log tmpi_flg=-1
        LOAD_CONV: Emission SSDU=18432
        LOAD_CONV: Reception SSDU=18432
        LOAD_CONV: Conversation Id=58960001
        INITIATE: State == S_INIT
```

```
Dumping For CMINIT() Conversation Table

Error Direction              : CM_RECEIVE_ERROR
Deallocation Type            : CM_INVALID_VALUE
Send Type                    : CM_SEND_AND_PREP_TO_RECEIVE
Conversation ID              : 58960001
Conversation Type            : CM_MAPPED_CONVERSATION
Mode                         : CM_INVALID_MODE
Vip Data Mode                : NOT TRANSPARENT
Status                       : S_INIT
Conversation Side            : INITIATOR_SIDE
Terminal Type                : DKU7107
Terminal Session Selector    : INVALID_VALUE
Emission SSDU                : 18432
Reception SSDU               : 18432
Tmpi Context Pointer         : 0
Application Session Selector: INVALID_VALUE
Presentation Type            : INVALID_VALUE
Correspondent Entity         : UNDEFINED_CORRESPONDENT
Local Entity                 : TERMIMAL

Dumping Submitter Structure.

USER     : USER
PASSWORD :
BILLING  :
PROJECT  :

Dumping Session Structures.

Session Selector Calling   : @API22678
Session Selector Called    : @JE60
Transport Selector Calling : @1JUC2
Transport Selector Called  : @1JUE6
Network Selector Calling   : 02608c2e5326
Network Selector Called    : 02608c2e5326

Dumping Transport Structures.

Transport Class              : 4
Alternative Transport Class  : 2
Use of Express Flow          : 1
Use of Flow Control          : 1
Transport Credit             : 2
Transport Provider Data Unit : 1024
Transport Checksum           : 0
Transport Type of Network    : 2

End of Data dumped.

Output Conversation ID = 58960001
RC: CM_OK

Mon Jan 31 16:20:02 1994
```

Chapter 3. Variables, Functions and Verbs

Summary

– Variables, on page 3-1.
– api_msg or rdmessg Function, on page 3-3.
– retrieve error Function, on page 3-5.
– CMACCP, on page 3-6.
– CMALLC, on page 3-9.
– CMDEAL, on page 3-11.
– CMINIT, on page 3-13.
– CMRCV, on page 3-16.
– CMSDT, on page 3-20.
– CMSED, on page 3-22.
– CMSEND, on page 3-24.
– CMSERR, on page 3-26.

General

This chapter is in three parts:

1. Explanation of the different variables that can be set to create this CPI-C SS product's operating environment.
2. Explanation of the *rdmessg* and *retrieve_error* functions.
3. List of the CPI-C SS verbs giving the information you need to use them.

Variables

The following environment variables can be set:

API_MODE

This variable can be set to the following values:

TRANSPARENT

NONTRANSPARENT

API_LOG_LEV

This variable contains the value of the trace level. The values range from 0 to 4, 0 is the minimum value. 0 is used to log the detected errors. The other values are reserved for authorized personnel.

API_LOG_FLG

This variable is used to select the output logging file. The values for this variable are:

0 to produce the logging data using the ELOG driver. The logging file is named **elgfile** under the directory /**usr**/**adm** and it is a circular file. **elgfile** is cleared at system startup. The command **lgprint** produces the file **elog_print** from the input file **elgfile**. **elog_print** can then be printed or displayed. It is recommended to use this value to produce the logging data. (Refer to the *OTM Administrator's and User's Guide* for further details about the command).

1 if the logging data is to be stored in a temporary file:

/tmp/apilog and can be viewed

using the command **cpi-clog**

This is not a fixed size file and can store a large quantity of logging information. This file is written in append mode. It is a temporary file and is therefore deleted at system start–up. To save the stored information, the temporary file must be copied into a private file before executing shutdown.

AUT_LOG_FILE

This variable contains the file name used to perform the start up procedure.

LOGNAME

This variable contains the user name who requests the connection. A check is performed before starting a connection.

API_SSDUE

It is used only with the API_TP_NAME variable. Its value is the size of the emission SSDU. The value ranges from 1024 bytes to 18432 bytes. If this variable is not set the CMACCP call uses the default value (18432 bytes).

API_SSDUR

It is used only with theAPI_TP_NAME variable. Its value is the size of the reception SSDU. The value ranges from 1024 bytes to 18432 bytes. If this variable is not set the CMACCP call uses the default value (18432 bytes).

api_msg or rdmessg Function

Purpose

This function gives you an analysis of the CPI-C SS error messages.

Syntax

char *api_msg (keymsg, myfile)

int keymsg;

char *myfile;

rdmessg

Description

To analyze the error messages that are produced by the CPI verbs, two functions are provided: *api_msg()* and *rdmessg*. These functions are not standard X/Open and are not CPI functions.

The error messages caused by the CPI verbs are collected in the file CPICMESSAGE.LIB under the directory */usr/cpi-c.* Each record in this file associates to each error code that is returned by the CPI functions, the correspondent meaning. The record format is the following:

message identifier (key)	4 digits
filler	3 spaces
message text	80 chars
end of string	1 byte

The file is ordered per key in ascending mode.

For a detailed list of the error messages and return codes, refer to the header file **<cpic-c.h>**.

The *api_msg()* function can be used in a C program. *keymsg* is the code of the error message. *myfile* is a pointer to the string containing the pathname of the file containing the errors. If this argument is NULL the pathname of the file CPICMESSAGE.LIB is assumed by default.

rdmessg can be used in a COBOL program. *keymsg* is the key that identifies the message *errmsg* is the output area which will contain the message found in the file. *rdmessg* calls the *api_msg()* function. The environment variable *COB_errmsg* contains the pathname of the file containing the errors. If this variable does not contain a value, the path of the CPICMESSAGE.LIB file is assumed by default.

Returns

The *api_msg()* function returns a pointer to the string containing the desired message.

Example

Here is an example of a COBOL program that uses *rdmessg*.

```
.
.
.
working-storage section.
01              keymsg  pic     s9(09)  comp.
01              errmsg.
        02      msg-id  pic     x(04).
        02      filler  pic     x(03).
        02      msg-txt pic     x(80).
        02      filler  pic     x(01).
procedure division.
.
.
call ”rdmessg” using            keymsg
                                errmsg
```

retrieve_error Function

Purpose

This function generates a message describing the last error encountered.

Syntax

retrieve_error

Description

This function is only effective if an error has occurred. It takes no parameters.

The *retrieve_error* function gets detailed diagnostic information from the Open Terminal Management (OTM) product which is the service provider. This information is used to generate a message.

This is an example of a message generated by the *retrieve_error* function:

```
*4644   2ltp load_conv: Entry XXXX not found in configuration
file  /usr/cpi-c/site.cnf
```

Example

This is an example of the use of *retrieve_error* in a C program:

```
if (returncode != CM_OK )

{

printf( \ nERROR DETECTED IN CALL TO <<%s>>\n\n, command);

printf( error : <<%s>>\n, api_msg( returncode, NULL));

printf( %s\n, retrieve_error());
```

CMACCP

Purpose

accept conversation

Syntax

C SYNTAX

```
#include ”cpi-c.h”
.
.
CMACCP (conversation_id, return_code)
CONVERSATION_ID conversation_id;
CM_RETCODE *return_code;
```

COBOL SYNTAX

```
 .
 .
COPY ”CPI–H”
 .
 .
CALL ”CMACCP” USING CONVERSATION-ID, CM-RETCODE
 .
```

Description

The *CMACCP* call accepts an incoming conversation (either from a remote host application or from a terminal application). Like *CMINIT*, *CMACCP* initializes values for various conversation characteristics. The difference between the two calls is that the program that will later allocate the conversation issues the *CMINIT* call, and the partner program that will accept the conversation after it is allocated issues the *CMACCP* call.

The following environment variables are to be set:

API_MODE	It can assume two different values: TRANSPARENT NON_TRANSPARENT
API_LOG_LEV	It contains the logging level value.
API_LOG_FLG	It contains the logging flag value.
LOGNAME	It is used only with the API_TP_NAME variable. It contains user and password.
API_TP_NAME	The cpi-c program is an host application: it can receive data from other terminal applications running only on DPX machines. The value of API_TP_NAME is used as local Session User (local Mailbox). (see LOCMB in the ”OTM Administrator’s and User’s Guide”).

API_TERM_NAME

The cpi-c program is a terminal application: it can receive data from other terminal applications running on GCOSx hosts or DPX or Datanet machines. The value of API_TERM_NAME corresponds to an address into the site.cnf file (Symbolic Destination Name).

API_TERM_NAME and API_TP_NAME must not be used together because they are in conflict.

When the *CMACCP* call completes successfully, the conversation characteristics are initialized as follows:

Conversation Characteristic	Initialized Value
conversation type	Derived from the received conversation start–up request
deallocate type	CM_DEALLOCATE_SYNC_LEVEL
prepare to receive type	CM_PREP_TO_RECEIVE_SYNC_LEV-EL
processing mode	CM_BLOCKING
receive type	CM_RECEIVE_AND_WAIT
return control	This characteristic applies only to an Al-locate call
send type	CM_BUFFER_DATA

Parameters

conversation_ID

[output] Specifies the conversation identifier assigned to the conversation. CPI–C supplies and maintains the *conversation ID*. When the *return code* is set equal to CM_OK the value returned in this parameter is used by the program on all subsequent calls issued for this conversation.

return_code

[output] Specifies the result of the call execution. The *return code* variable can have one of the following values:

CM_OK

CM_PROGRAM_STATE_CHECK

Indicates failure in the node service environment that provides the synchronization between the program (in which the *CMACCP* call is issued) and the remote request.

CM_PROGRAM_PARAMETER_CHECK

Reflects one of the following situations:

- API_TERM_NAME variable is set and does not refer any entry into *site.cnf* file.
- API_TERM_NAME and API_TP_NAME variables are both set.
- API_TERM_NAME variable is set and its length is greater than eight characters.
- API_TP_NAME variable is set and its length is greater than twelve characters.
- Neither API_TERM_NAME nor API_TP_NAME variables are set.
- API_TP_NAME variable is set and LOGNAME one is unset.
- API_TP_NAME variable is set and LOGNAME variable is greater than twelve.
- API_TP_NAME variable is set, LOGNAME variable too, but password value could not be extracted from LOGNAME.

CM_RESOURCE_FAILURE_ RETRY

The error condition must be removed before attempting a retry execution.

State Changes

When return code is set to CM_OK, the conversation enters the Receive state.

Implementation Specifics

For each conversation, CPI-C assigns a unique identifier (the conversation ID) that the program uses in all future calls intended for that conversation. Therefore, the program must issue the CMACCP call before any other calls can refer to the conversation.

This call is used when the DPX machine receives an incoming request to establish a session connection.

When the remote session request comes in, the following operations are executed:

1. The check on the validity of this parameter is executed: LOGNAME
2. In case of positive result of the checks the local program is scheduled and it begins a dialogue confirming the connection request. Otherwise the session connection is rejected with an error code.
3. Therefore, when CM OK is delivered the conversation is accepted and no synchronization call is required.

CMALLC

Purpose

allocate

Syntax

C SYNTAX

```
#include ”cpi-c.h”
CMALLC (conversation_id, return_code)
CONVERSATION_ID conversation_id;

CM_RETCODE *return_code;
```

COBOL SYNTAX

```
.
.
.
COPY ”CPI-C”
.
.
.
CALL ”CMALLC” USING CONVERSATION–ID, CM-RETCODE
.
.
```

Description

A program uses the *CMALLC* call to establish a mapped conversation (depending on the conversation type characteristics) with its partner program.

After a *CMALLC* has been executed, the conversation is always in Send state for the CPI-C initiator side.

A return code of CM_OK indicates that the conversation has been successfully allocated.

Parameters

conversation_ID
: [input] Specifies the conversation identifier of an initialized conversation.

return_code
: [output] Specifies the result of the call execution, which is returned to the local program. The **return control** characteristic determines which *return codes* can be returned to the local program. If **return control** is set to CM_WHEN_SESSION_ALLOCATED, *return code* can have one of the following values:

CM_OK

CM_ALLOCATE_FAILURE_RETRY

The error condition must be removed before attempting a retry execution.

CM_PARAMETER_ERROR

This value indicates that an error has been detected on the information placed in the **configuration file** or a mismatching between this information and lower layer configuration has been detected.

CM_PROGRAM_STATE_CHECK

This value indicates that the conversation is not in Initialize state.

CM_PROGRAM_PARAMETER_CHECK

This value indicates that the *conversation ID* specifies an unassigned conversation identifier.

State Changes

Conversation enters Send state (if *Transparent Mode* is not enabled).

In *Transparent Mod*e the connection state is returned (this state cannot be conform to the CPI-C rules).

Implementation Specifics

1. An allocation error resulting from the local system's failure to obtain a session for the conversation is reported on the *CMALLC* call. An allocation error resulting from the remote correspondent rejection of the allocation request is reported on *CMALLC*.
2. To establish the conversation for CPI-C, a session must first be established between the local correspondent and the remote correspondent.
3. The request of session/conversation, towards the correspondent is performed when the CMALLC call is requested.
4. In Transparent Mode *CMALLC* consists of opening the connection with a host. In this case, the return code is RECEIVE (in CPI-C the return code is SEND).

If the correspondent is GCOS 8 the login procedure is to be performed by the user application.

CMDEAL

Purpose

deallocate

Syntax

C SYNTAX

```
#include ”cpi-c.h”
CMDEAL (conversation_id, return_code)
CONVERSATION_ID conversation_id;
CM_RETCODE *return_code;
```

COBOL SYNTAX

```
.
.
COPY ”CPI–H”
.
.
.
CALL ”CMDEAL” USING CONVERSATION-ID, CM–RETCODE
```

Description

A program uses the *CMDEAL* call to end a conversation. The *conversation_ID* is no longer assigned when the *CMDEAL* call completes successfully.

In GCOSx Host connections, *CMDEAL* always consists of a connection abort and for this reason **deallocate type** is always set to CM_DEALLOCATE_ABEND.

In UNIX to UNIX connections, *CMDEAL* must be executed in Send state. Executing a *CMDEAL* not in Send state causes a connection abort and for this reason **deallocate type** is set to CM_DEALLOCATE_ABEND.

When the conversation is in *SEND* state the *CMDEAL* call completes successfully and the partner program will receive a CM_DEALLOCATE_NORMAL.

Parameters

conversation_ID

[input] Specifies the conversation identifier of the conversation to be ended.

return_code

[output] Specifies the result of the call execution, which is returned to the local program.

If **deallocate type** is set CM_DEALLOCATE_ABEND (always set to this value), the *return_code* variable can have one of the following values:

CM_OK
(deallocation is complete)

CM_PROGRAM_STATE_CHECK

CM_PROGRAM_PARAMETER_CHECK
This value indicates that the conversation specifies an unassigned *conversation_ID* identifier.

State Changes

When return code indicates CM_OK, the conversation enters Reset state.

CMINIT

Purpose

initialize conversation

Syntax

C SYNTAX

```
#include ”cpi-c.h”
CMINIT     (conversation_id, system_dest_name, return_code)
CONVERSATION_ID conversation_id;
char *system_dest_name;
CM_RETCODE *return_code;
```

COBOL SYNTAX

```
.
.
.
COPY ”CPI–H”
.
.
CALL ”CMINIT” USING CONVERSATION–ID, SYSTEM-DEST-NAME, CM-RETCODE
.
```

Description

A program uses the *CMINIT* call to initialize values for various conversation characteristics before allocating the conversation (with a call to *CMALLC*).

The following environment variables can be set:

API_MODE

It can assume the following values:

TRANSPARENT

NON_TRANSPARENT

In a DPX environment this variable is to be set to TRANSPARENT.

AUT_LOG_FILE

This variable contains the path of a command file used to perform the conversation after the connection. The command file contains ”send” and ”receive” commands. The last command must be a ”receive” command.

If the variable API_MODE is set toTRANSPARENT this variable must not be used.

USERINFO

This variable cannot exceed 32 chars. It invokes the DSA functions.

The information contained in this variable will be sent to the application running on Host and will be used by the Host.

SECURITY

INITIAL_COR

EMU_MODEL

The variables USERINFO, SECURITY, INITIAL_COR, EMU_MODEL are used by the Terminal Manager. Their values are inserted in the connection protocol record.

LOCMB It contains the value of the Local Session Selector (Local Mailbox).

REMMB_EXT

It contains the value of the Remote Session Selector Extension (remote mailbox extension). This variable is used for security.

API_LOG_LEV

It contains the trace level value.

API_LOG_FLG

It contains the trace storage mode flag value. When the CMINIT call completes successfully, the following conversation characteristics are initialized:

Conversation Characteristic	Initialized Value
conversation type	CM_MAPPED_CONVERSATION
deallocate type	CM_DEALLOCATE_ABEND
prepare to receive type	CM_PREP_TO_RECEIVE_SYNC_LEVEL
processing mode	CM_BLOCKING
receive type	CM_RECEIVE_AND_WAIT
return control	CM_WHEN_SESSION_ALLOCATED
send type	CM_SEND_AND_PREP_TO_RECEIVE

Parameters

conversation_ID

[output] Specifies the conversation identifier assigned to the conversation, which is returned to the program. CPI C supplies and maintains the *conversation_ID*. If the *CMINIT* is successful (*return_code* is set equal to CM_OK), the local program uses the identifier returned in this variable for the rest of the conversation.

system_dest_name

[input] Specifies the symbolic name of the correspondent for the session on which the conversation is to be performed. The symbolic destination name is provided by the program and refers to an entry in the **configuration file**. The appropriate entry in the **configuration file** is retrieved and used to initialize the characteristics for the conversation being initialized.

return_code

[output] Specifies the result of the call execution, which is returned to the local program. The *return_code* variable can have one of the following values:

CM_OK

CM_PROGRAM_PARAMETER_CHECK
This value indicates that the *system_dest_name* does not match an entry in the */usr/cpi-c/site.cnf* file.

State Changes

When return code indicates CM_OK, the new conversation is in the Initialize state.

Implementation Specifics

1. For each conversation, CPI–C assigns a unique identifier, the *conversation_ID*. The program then uses the *conversation_ID* in all future calls intended for that conversation. *CMINIT* must be issued by the program before any other calls may be made for that conversation.
2. If the symbolic destination name contains invalid allocation information, the error is detected when the information is processed by *CMALLC*.

CMRCV

Purpose

receive information

Syntax

C SYNTAX

```
#include ”cpi-c.h”
CMRCV    (conversation_id, buffer, requested_length,
          data_received, received_length, status_received,
          request_to_send_received, return_code)
CONVERSATION_ID conversation_id;
char *buffer;
int *request_length;
DATA_RECEIVED *data_received;
int *received_length;
STATUS_RECEIVED *status_received;
REQUEST_TO_SEND_RECEIVED *request_to_send_received;
CM_RETCODE *return_code;
```

COBOL SYNTAX

```
.
.
COPY ”CPI–H”
.
.
.
CALL ”CMRCV” USING CONVERSATION–ID, CM–BUFFER,
CM–REQUEST–LENGTH, DATA–RECEIVED,
CM–RECEIVED–LENGTH, STATUS–RECEIVED,
REQUEST–TO–SEND–RECEIVED, CM–RETCODE
```

Description

A program uses the *CMRCV* call to receive information from a given conversation. The information received can be a data record (on a mapped conversation), conversation status.

Parameters

conversation_ID

[input] Specifies the conversation identifier.

buffer

[input] Specifies the variable in which the program has to receive data.

If *data_received* is returned to the program with a value of CM_NO_DATA_RECEIVED, or *return_code* has a value other than CM_OK or CM_DEALLOCATED_NORMAL, buffer contents are undefined.

requested_length

[input] Specifies the maximum amount of data the program has to receive (SPDU).

data_received

[output] Specifies whether or not the program received data, which is returned to the local program.

Unless *return code* is set to CM_OK or CM_DEALLOCATED_NORMAL, *data_received* is undefined. The *data_received* variable can have one of the following values:

CM_NO_DATA_RECEIVED
(mapped conversation)
No data is received by the program. Status may be received if the *return_code* is set to CM_OK.

CM_COMPLETE_DATA_RECEIVED
(mapped conversations)
For mapped conversations, indicates that a complete data record has been received.

received_length

[output] Specifies the variable in which is returned the amount of data the program received, up to the maximum.

If the program receives information other than data, *received_length* is undefined.

status_received

[output] Specifies the variable in which is returned an indication of whether or not the program received the conversation status.

Unless *return_code* is set to CM_OK, *status_received* is undefined.

The *status_received* variable can have one of the following values:

CM_NO_STATUS_RECEIVED
No conversation status is received by the program; data may be received.

CM_SEND_RECEIVED
The remote side of the conversation has entered Receive state, placing the local side in Send state. The local program (who issued the *CMRCV* call) can now issue *CMSEND*.

request_to_send_received

[output] Specifies the variable in which is returned an indication of whether or not the remote program issued a Request To Send call.

request to send received is undefined when return code is CM_PROGRAM_PARAMETER_CHECK or CM_PROGRAM_STATE_CHECK.

The *request_to_send_received* variable can have the following value:

CM_REQ_TO_SEND_NOT_RECEIVED

The local program has not received a request–to–send notification.

return_code

[output] Specifies the result of the call execution, which is returned to the local program. The return codes that can be returned depend on the state and characteristics of the conversation at the time this call is issued.

If *CMRCV* is issued in Send state, *return_code* can have one of the following values:

CM_OK

CM_ DEALLOCATED_ABEND

CM_ DEALLOCATED_NORMAL

CM_ RESOURCE_FAILURE_RETRY

CM_ PROGRAM_ERROR_TRUNC

CM_ PROGRAM_ERROR_PURGING

If a state or parameter error has occurred, return code can have one of the following values:

CM_PROGRAM_STATE_CHECK
This value indicates one of the following:
The conversation is not in Send, or Receive state.

CM_PROGRAM_PARAMETER_CHECK
This value indicates one of the following:
The *conversation_ID* specifies an unassigned conversation identifier.
The *requested_length* specifies a value not able to contain the received data.

State Changes

When return code indicates CM_OK:

The conversation is in Receive state if a *CMRCV* call is issued and all of the following conditions are true:

data_received indicates CM_COMPLETE_DATA_RECEIVED.

The *status_received* indicates CM_NO_STATUS_RECEIVED.

The conversation enters Send state when *status_received* is set to CM_SEND_RECEIVED.

No state change occurs when the call is issued in Receive state, *data_received* is set to CM_COMPLETE_DATA_RECEIVED, and *status_received* indicates CM_NO_STATUS_RECEIVED.

The state change is not implicit at the end of the operation, as for a *CMSEND* operation, one or more *CMRCV* can be performed.

Implementation Specifics

1. As **receive type** is set to CM_RECEIVE_AND_WAIT, processing mode is set to CM_BLOCKING, and no data is present when the call is performed, CPI–C waits for information to arrive on the specified conversation before allowing the *CMRCV* call to return with the information. If information is already available, the program receives it without waiting.
2. If the return code indicates CM_PROGRAM_STATE_CHECK or CM_PROGRAM_PARAMETER_CHECK, all other variables will contain no information.
3. A Receive call issued against a mapped conversation can receive only a complete data record.

   When the program receives a complete data record, the *data_received* parameter is set to CM_COMPLETE_DATA_RECEIVED. The length of the record is less than or equal to the length specified on the requested length parameter.
4. The *CMRCV* call performed with requested length set to zero has no special significance. The type of information available is indicated by the *return_code*, *data_received*, and the *status_received* parameters, as usual. As **receive type** is set to CM_RECEIVE_AND_WAIT and no information is available, this call waits for information to arrive.
5. The program can receive both data and conversation status on the same call. The *return_code*, *data_received*, and *status_received* parameters indicate to the program the kind of information the program receives.

CMSDT

Purpose

set deallocate type

Syntax

C SYNTAX

```
#include ”cpi-c.h”
CMSDT (conversation_id, deallocate_type, return_code)
CONVERSATION_ID conversation_id;
DEALLOCATE_TYPE *deallocate_type;
CM_RETCODE *return_code;
```

COBOL SYNTAX

```
.
.
.
CALL CMSDT USING CONVERSATION–ID,
CM-DEALLOCATE-TYPE, CM-RETCODE
```

Description

CMSDT is used by a program to set the deallocate type characteristic for a given conversation.

CMSDT overrides the value that was assigned when either the *CMINIT* or the *CMACCP* call was issued.

Parameters

conversation_ID

[input] Specifies the conversation identifier.

deallocate_type

[input] Specifies the type of deallocation to be performed. The deallocate type variable can have one of the following values:

CM_DEALLOCATE_FLUSH

The conversation is deallocated normally.

CM_DEALLOCATE_ABEND

Execute the function of the Flush call when the conversation is in Send state and deallocate the conversation abnormally. Data purging can occur when the conversation is in Receive state.

return_code

[output] Specifies the result of the call execution, which is returned to the local program. The return code variable can have one of the following values:

CM_OK

CM_PROGRAM_PARAMETER_CHECK

This value indicates one of the following:

The conversation ID specifies an unassigned conversation identifier.

The deallocate type specifies an undefined value.

CM_PROGRAM_STATE_CHECK

State Changes

This call does not cause any state changes.

Implementation Specifics

1. A deallocate type set to CM_DEALLOCATE_FLUSH is used by a program to deallocate a conversation normally.
2. A deallocate type set to CM_DEALLOCATE_ABEND is used by a program to unconditionally deallocate a conversation. Specially, the parameter is used when the program detects an error condition that prevents further useful communications (communications that would lead to successful completion of the transaction).
3. If a return code other than CM_OK is returned on the call, the deallocate type conversation characteristic is unchanged.

CMSED

Purpose

set error direction

Syntax

C SYNTAX

```
#include ”cpi-c.h”
.
.
CMSED (conversation_id, &error_direction, &return_code)
CONVERSATION_ID conversation_id;
ERROR_DIRECTION *error_direction;
CM_RETCODE *return_code;
```

COBOL SYNTAX

```
.
COPY ”CPI-H”
.
CALL ”CMSED” USING CONVERSATION–ID,
    ERROR–DIRECTION, CM–RETCODE
.
```

Description

This call is used by the program to set the error direction characteristic for a given conversation, overriding the value assigned when CMINIT/*CMACCP* calls were issued. This call is useful only for DPX/DPX connections and is ignored for GCOSx connections.

Parameters

conversation_ID

[input] specifies the conversation identifier

error_direction

[input] specifies the error direction of the data flow in which the program detected an error.

The error direction can have the following values:

CM_RECEIVE_ERROR

specifies that the program detected an error in the data it received from the remote program.

CM_SEND_ERROR

specifies that the program detected an error while preparing to send data to the remote program.

return_code

[output] can have one of the following values:

CM_OK

CM_PROGRAM_PARAMETER_CHECK

conversation ID or error direction wrong

CM_PROGRAM_STATE_CHECK

State Changes

This call does not cause any state change

Implementation Specifics

1. The error direction is significant only if *CMSERR* is issued after a RECEIVE on which both data and CM_SEND_RECEIVED are received. Otherwise the *error_direction* is ignored when this call is issued. In this situation, the *CMSERR* may result from one of the following errors:

   an error in the received data

   an error being the result of processing performed after having received and processed the data

   Because the correspondent in this situation cannot tell which error occurred, the program has to supply the error direction information.

   CM_RECEIVE_ERROR is the default value, that can be overridded by *CMSED* before issuing *CMSERR*. Once changed, the new *error_direction* remain in effect until the program change it again.

2. If a *return_code* other than CM OK is returned on the call, the *error_direction* characteristics is unchanged.

CMSEND

Purpose

send data

Syntax

C SYNTAX

```
#include ”cpi-c.h”
CMSEND  (conversation_id, buffer, send_length,
             request_to_send_received, return_code)
CONVERSATION_ID conversation_id;
char *buffer;
int *send_length;
REQUEST_TO_SEND_RECEIVED *request_to_send_received;
CM_RETCODE *return_code;
```

COBOL SYNTAX

```
.
.
.
COPY ”CPI–H”
.
.
.
CALL ”CMSEND” USING CONVERSATION–ID, CM_BUFFER,
   CM–SEND–LENGTH, REQUEST–TO–SEND–RECEIVED,
   CM–RETCODE
.
.
```

Description

A program uses the *CMSEND* call to send data to the remote program. This call transfers one data record to the correspondent for transmission to the remote program. The data record consists entirely of data and is not examined by the correspondent for possible logical records.

Parameters

Conversation_ID

[input] Specifies the conversation identifier of the conversation.

buffer

[input] When a program issues a *CMSEND* call, buffer specifies the data record to be sent. The length of the data record is given by the send length parameter.

send_length

[input] Specifies the message length. The send length ranges in value from 0 to the SSDU value and it depends on the correspondent.

It specifies the size of the buffer parameter and the number of bytes to be sent during the conversation. When a program issues a *CMSEND* call during a mapped conversation and send length is zero, a null data record is sent.

request_to_send_received

[output] Specifies the variable in which is returned an indication of whether or not a request send notification has been received. The request send received variable can have the following value:

CM_REQ_TO_SEND_NOT_RECEIVED

Indicates that a request–to–send notification has not been received.

If return code is either

CM_PROGRAM_STATE_CHECK, or

CM_PROGRAM_PARAMETER_CHECK, request send received is undefined.

return_code

[output] Specifies the result of the call execution which is returned to the local program. The return code variable can have one of the following values:

CM_OK

CM_DEALLOCATED_ABEND

CM_RESOURCE_FAILURE_NO_RETRY

CM_PROGRAM_ERROR_PURGING

CM_PROGRAM_STATE_CHECK This value indicates that the conversation is not in Send state.

CM_PROGRAM_PARAMETER_CHECK This value indicates that the *conversation_ID* specifies an unassigned conversation identifier.

State Changes

When *return_code* indicates CM_OK:

The conversation enters Receive state when *CMSEND* is issued with send set to CM_SEND_AND_PREP_TO_RECEIVE.

Implementation Specifics

1. The local correspondent sends immediately the data for the transmission.
2. The *CMSEND* call sends one complete data record. No control is performed on sent data as mapped conversation mode is used.
3. In Transparent Mode, no control is performed on the state when the command is executed.

   The connection rules and the remote rules are applied.

CMSERR

Purpose

send error

Syntax

C SYNTAX

```
#include ”cpi-c.h”
CMSERR  (conversation_id, request_to_send_received,
            return_code)
CONVERSATION_ID conversation_id;
REQUEST_TO_SEND_RECEIVED *request_to_send_received;
CM_RETCODE *return_code;
```

COBOL SYNTAX

```
.
.
COPY ”CPI–H”  .
.
.
CALL ”CMSERR” USING CONVERSATION–ID, REQUEST–TO–SEND–RECEIVED,
CM–RETCODE
```

Description

CMSERR is used by a program to inform the remote program that the local program detected an error during a conversation. If the conversation is in Send state, *CMSERR* forces the correspondent to purge its send buffer. When this call completes successfully, the local program is in Send state and the remote program is in Receive state. Further action is defined by program logic.

Parameters

conversation_ID

[input] Specifies the conversation identifier.

request_to_send_received

[output] Specifies the variable in which is returned an indication of whether or not a request–to–send notification has been received. The *request_to_send_received* variable can have one of the following values:

CM_REQ_TO_SEND_RECEIVED

The local side of the conversation enters Receive state, which places the remote side in Send state. (for GCOSx *request_to_send_received* is always set to CM_REQ_TO_SEND_RECEIVED.)

CM_REQ_TO_SEND_NOT_RECEIVED

A request-to-send notification has not been received.

If *return_code* is set to either CM_PROGRAM_PARAMETER_CHECK or CM_PROGRAM_STATE_CHECK, the value of *request_to_send_received* is undefined.

return_code

Specifies the result of the call execution, which is returned to the local program. The value for return code depends on the state of the conversation at the time this call is issued.

If the *CMSERR* is issued in Send state, *return_code* can have the following value:

CM_OK

If the *CMSERR* is issued in Receive state, *return_code* can have one of the following values:

CM_OK

CM_DEALLOCATED_NORMAL

CM_RESOURCE_FAILURE_NO_RETRY

State Changes

When return code indicates CM_OK:

The conversation enters Send state when the call is issued in Receive state.

No state change occurs when the call is issued in Send state.

Implementation Specifics

1. The correspondent sends the error notification to the remote correspondent immediately (during the processing of this call).
2. When *CMSERR* is issued in Receive state, incoming information is also purged. Because of this purging, the return code of CM_DEALLOCATED_NORMAL is reported instead of:

   CM_DEALLOCATED_ABEND

   Similarly, a *return_code* of CM_OK is reported instead of:

   CM_PROGRAM_ERROR_NO_TRUNC

   CM_PROGRAM_ERROR_PURGING

   The following types of incoming information are also purged:

   Data sent with the Send Data call

   Deallocate calls

   The request-to-send notification is not purged. This notification is reported to the program when it issues a call that includes the *request_to_send_received* parameter.
3. A *CMSERR* sent to a GCOSx Host corresponds to the BREAK functionality particular to the Host.

   After the execution of this command, the local application is in a RECEIVE status, specified by the parameter request send received.

   This occurs to match *CMSERR* rules and GCOSx rules.

Appendix A. Error Messages, Return Codes and Trace

Summary

- Error Messages, on page A-1.
- Return Codes, on page A-4.
- CPI-C SS Trace Facility, on page A-5.

Error Messages

Legend

The error messages generated by CPI-C Starter Set are listed with a particular structure.

Within the text of the messages, the following identifiers have been used:

xxxx	to indicate an hexadecimal code
yyyy	to indicate a printer name or an alphanumeric string
zzzz	to indicate a decimal number
id–num	to identify the connection number.

Some error messages contain the following sentence:

return code xxxx cause xxxx orig xxxx

where:

return code	is the error code from the layer
orig	is the number of the layer from which the error occurred
cause	is the reason of the error

The command

```
pmaderror retcode
```

or

```
pmaderror <orig> <cause>
```

displays on the screen a description of the error code.

For a further description of the meaning of return code, origin and cause, refer to the ”ISO Services Manual”.

CPI-C Starter Set Configurator Messages

0200 2LTP configurator is already in use.

1. The 2LP configurator is being updated by another user.
2. Wait for the termination of the configurator.

0201 Internal error zzzz

1. A fatal error has occured.
2. Call Technical Assistance.

0202 2LTP configurator: end of function

1. The CPI-C Starter Set configuration session has finished.
2. Information only.

0203 File reorganization failed. Status = zzzz.

1. An error in the record reorganization in the site.cnf file has occurred.

0204 Ck_dest (): Cannot Open configuration file. Status = zzzz

1. The site.new file does not exist or it is damaged.

0205 Ck_dest() corrupted configuration file. Status = zzzz. Please use reorg().

1. The site.new file contains damaged records.

0206 Ck_dest() unexpected end of file. Status = zzzz.

1. An error reading /usr/CPI-C/site.new file has occurred.

0207 Crtcnf() cannot open configuration file. Status = zzzz.

1. An error reading /usr/CPI-C/site.new file has occurred.

0208 Crtcnf() cannot write data on file. Status = zzzz

1. An error writing site.new file has occurred.

0209 Dltcnf() cannot open configuration file. Status = zzzz.

1. An error writing /usr/CPI-C/site.new file has occurred.

0210 Dltcnf() cannot open temporary file. Status = zzzz.

1. The temporary file cannot be read under /tmp directory.
2. Check access rights.

0211 Dltcnf() cannot write data on temporary file. Status = zzzz.

1. The temporary file cannot be created under /tmp directory.
2. Check write rights.

0212 Dltcnf() unexpected end of file on configuration file. Status = zzzz.

1. A UNIX system call error has occurred.
2. Call Technical Assistance.

0213 Lstcnf() cannot open configuration file. Status = zzzz.

1. The configuration file cannot be read.
2. Check access rights.

0214 Lstcnf() cannot open temporary file. Status = zzzz.

1. The temporary file cannot be read under /tmp directory.
2. Check access rights.

0215 Lstcnf() unexpected end of file on configuration file. Status = zzzz.

1. An UNIX system call error has occured.
2. Call Technical Assistance.

0216 Save_file() cannot save configuration file. Status = zzzz.

1. A copy of the configuration file cannot be created under /tmp directory.
2. Check access rights.

0217	Save_file() cannot execute system() function. Status = zzzz. 1. An UNIX system call error has occured. 2. Call Technical Assistance.
0218	Restore_file() cannot restore the configuration file. Status = zzzz. 1. The file site.cnf cannot be created under /usr/CPI-C directory. 2. Check access rights.
0219	Delete_file() cannot remove copy file. Status = zzzz. 1. It is not possible to remove the copy of the configuration file under /tmp. 2. Check access rights.
0220	Exit_fun() Signal received. Number = zzzz. 1. The configurator may be killed because of a received signal. 2. Call Technical Assistance.
0221	Crtcnf() cannot open configuration file in write mode. Status = zzzz. 1. The configuration file cannot be written under /usr/CPI-C directory. 2. Check write rights.
0222	Create_file() cannot create configuration file. Status = zzzz. 1. It is not possible to create site.new file for the first time. 2. Check access rights.
0223	Create_file() cannot remove configuration file for errors. Status = zzzz. 1. It is not possible to delete site.new file. 2. Check access rights.
0224	Ck_dest() duplicated entries in configuration file. Status = zzzz. 1. Two equal entries have been found in the configuration file. 2. Check the configured sites.
0227	Conf_stid: usage allowed only for super–user 1. Only super–user can launch the command.

Return Codes

The following return codes are provided for the CPI-C functions in BULL environment:

CM_OK	0
CM_ALLOCATE_FAILURE_NO_RETRY	1
CM_CONVERSATION_TYPE_MISMATCH	3
CM_TP_NOT_RECOGNISED	9
CM_TP_NOT_AVAILABLE_NO_RETRY	10
CM_DEALLOCATED_ABEND	17
CM_DEALLOCATED_NORMAL	18
CM_PARAMETER_ERROR	19
CM_PRODUCT_SPECIFIC_ERROR	20
CM_PROGRAM_ERROR_NO_TRUNC	21
CM_PROGRAM_ERROR_PURGING	22
CM_PROGRAM_PARAMETER_CHECK	24
CM_PROGRAM_STATE_CHECK	25
CM_RESOURCE_FAILURE_NO_RETRY	26
CM_DEALLOCATED_ABEND_TIMER	31

CPI-C SS Trace Facility

CPI-C Starter Set provides an integrated mechanism to trace events about data structures. Data is saved into a disk file that will be edited through a specific utility.

All the detected errors are collected in the file **/tmp/api.trc**. Every error message has the following format:

```
Pid = Pidid - Date Time Year

ernum <module>: <message>
```

where:

ernum	is the message code number.
module	is the internal routine that detected the error.
message	is a brief explanation of the error code.

An example follows:

```
Pid = 11054 - Thu Oct 14 11:30:20 1993

4643 21tp load_conv: Cannot open configuration file
/usr/CPI-C/site.cnf
```

CPI-C Starter Set manages two different ways to produce the output logging file:

1. via a stream driver, the ELOG driver (which provides minimum system interferences). Logging data is produced on a circular file (elgfile) under the directory /**usr**/**adm** on disk. **elgfile** is cleared at system startup.

   The command **lgprint** produces the file **elog_print** from the input file **elgfile**. **elog_print** can then be printed or displayed.

2. via private file containing all data collected from the logging activation.

   Logging information is stored into the sequential file /**tmp**/**apilog** and can be viewed using the command **cpi-clog** and **dumpcpic** (Refer to the OTM Administrator's and User's Guide for further details).

   utmlog can also be used to inspect the logging file. For example:

   ```
   utmlog -F/tmp/apilog -i AD > outfile
   ```

   to produce the *outfile* file from the input file */tmp/apilog*. *outfile* contains information about the function identifier *AD*. Refer to the **utmlog** command in the OTM Administrator and User's Guide.

Trace Levels

The trace level contains the following information:

Level 0	Error logging (recommended value) is traced in /tmp/apilog. This file can be viewed using the command **cpi-clog**.
Level 2	State and events of the CPI-C Starter Set. Function names and their parameters and function return codes are traced. The logging file can be inspected using the commands **cpi-log** and **dumpcpic**.
Level 4	All internal routines are traced. The logging file can be inspected using the commands **cpi-clog** and **dumpcpic**.

Each level includes the lower ones: a trace level 4 includes information from the level 0, 2.

For troubleshooting it is possible to raise the trace level: set the trace level to 4 and refer to the Technical Staff. If the trace level is 4, all the errors and the internal routine are logged.

The other values (5–15) are reserved for future use.

Trace Settings

The Trace Level and Trace Storage Mode flags can be set as follows:

1. via smit using the smit menus: ”Communications Applications and Services”, ”OTM Configuration”, ”Environment Configuration”, ”Change/Show Global Parameter”

   Refer to the ”Configuration” chapter for the procedure to be followed.

   These values are configured and are active for all the running OTM processes (provider, daemon and TM users: CPI-C Starter Set).

2. via environment variables.

   The environment variables are used to set a specific value of trace level and trace storage mode flag for a particular TM user (for example CPI-C Starter Set) to be launched and tested.

   CPI-C Starter Set trace level and trace storage mode flag can also be set using environment variables:

   API_LOG_LEV to set the trace level

   Specify a value ranging from 0 to 15.

   API_LOG_FLG to set the trace storage mode flag and the subsystem through which data is to be redirected

   Specify a value between 0 and 1.

   the variables affect only the CPI-C Starter Set environment.

3. via smit using the smit menus: ”Communications Applications and Services”, ”OTM Configuration”, ”CPI-C Starter Set Configuration”, ”Insert Symbolic Description”

   Trace level and trace storage mode flag set via ”CPI-C Starter Set Configuration” has the highest priority among the environment variables and the OTM trace level and flag.

Glossary

2LP
Two-Level Transaction Processing describes communications between two systems that do not have equal rights. One is considered the master, the other the slave.

AFNOR
Association Francaise de NORmalisation: French Standards Association.

API
Application Programming Interface: Functional interface allowing a high–level language application program to use specific data or functions of the operating system.

ASCII
American National Standard Code for Information Interchange.

AIX
International Business Machines UNIX Operating System derived from AT&T UNIX System V..The Bull DPX/20 uses this operating system.

C Language
The programming language that is the basis of the UNIX operating system.

CCITT
Consultative Committee on International Telegraphy & Telephone: United Nations Specialized Standards Group proposing recommendations for international telecommunications.

CEN/CENELEC
Comité Européen de Normalisation ELECtronique: European Electronic Standards Committee.

CICS
The Customer Information Control System is the IBM transaction monitor which is an interface with IBM mainframes.

COBOL (Micro Focus)
The Common Business Oriented Language from Micro Focus ltd. is the version used on the Bull UNIX computers.

Conversation ID
This is the unique identifier assigned by CPI-C SS to each conversation between a Bull UNIX machine and a Bull or IBM mainframe

CPI-C
Common Programming Interface for Communications: API allowing X/Open-compliant systems to communicate with systems implementing XCP2 protocols and SNA Logical Unit type 6.2 (LU6.2).

CSMA–CD
Carrier Sense Multiple Access – Collision Detection.

CXI
The Common Exchange Interface is the GCOS 8 module that provides the link to a Datanet.

DAC
The Direct ACcess module of GCOS 8 that manages terminal connections.

Datanet
The Bull communications processor, also known as a Front-end processor. It manages network communications.

DMVITP
Data Manager IV for Transaction Processing is a part of the GCOS 6 and GCOS 8 operating systems.

DPS
Distributed Processing System is the name given to the Bull mainframe computers (DPS 6000, DPS 7000 and DPS 9000)

DPX
A name given to Bull UNIX computers.

DSA
Distributed Systems Architecture is the Bull network architecture.

DTF
A GCOS 6 transaction processing executive.

ECL
A transaction processing module used by GCOS 6.

Environment Variable
These variables set up the environment for running an application or using a computer system under UNIX.

ECMA
European Computer Manufacturers' Association.

Ethernet
A baseband LAN specification (IEEE 802.3) using the CSMA-CD technique.

FastPath
Simplified keystroke commands permitting SMIT functions to be quickly activated (IBM).

FDDI
Communications adapter interface with a Fiber Distributed Data Network.

Gateway
Software, linking two networks using different communication architectures. A gateway performs routing, conversion and relaying operations.

GCOS
The General Comprehensive Operating Systems are the proprietary operating systems for the Bull mainframe computers: GCOS 6 for the DPS 6000, GCOS 7 for the DPS 7000, and GCOS 8 for the DPS 9000.

HDLC
High–level Data Link Control: Use of specialized series of bits to control data links in accordance with International Standards.

HPAD
Host PAD: Server side in the PAD client/server model.

IEEE
Institute of Electrical & Electronic Engineers.

IOF
The Interactive Operator Facility controls interactive processing under GCOS 7.

ISDN
Integrated Services Digital Network: Network supporting voice and non–voice communications.

ISO
International Standards Organization: Originator of Open Systems Interconnection reference model (ISO–IS 7498).

Mandatory
Characteristic of a parameter field. If data is not entered in the field, the command of the dialog is not executed (SMIT).

Mapped Conversation
In a mapped conversation there is a one-to-one correlation between the data record sent and the data record received.

MASK
A pattern of characters used as a control for other patterns of characters.

OSF
The Open Systems Facility is the gateway in the Bull Datanet that provides the interface to the IBM mainframes.

OSI
Open Systems Interconnection: Reference model defined in OS–IS 7498.

OTM
The Open Terminal Manager product provides the emulations needed for the Bull UNIX machines to communicate with Bull and IBM mainframes.

Output
Window where the results of dialog commands are displayed. The standard output of commands are sent to this window (SMIT).

PAD
Packet Assembler Disassembler: Functional device enabling un–equipped Data Terminal Equipments to access a packet switching network.

PDU
Protocol Data Unit: Unit of protocol control information specified in the protocol of a given layer.

PID
The French acronym (Prise Iso Dsa) for the ISO DSA Plug which provides conversion between ISO and DSA protocols.

Session
Session protocol: Virtual relationship permitting communications between two network addressable units.

SMIT
System Management Interface Tool (IBM): Menu–driven, resident command–building system management facility.

SMTP
Shared Memory Transport Protocol.

SNA
Systems Network Architecture is the IBM network architecture.

SPDU
The Session Protocol Data Unit is the data entering or leaving the bottom of the session layer.

SSDU
The Session Service Data Unit is the data entering or leaving the top of the session layer.

TDS
The Transaction Driven System is the GCOS 7 facility that handles transaction processing.

TP
This is the abbreviation for Transaction Processing.

TPAD
Terminal PAD: Client side in the PAD client/server model.

TPS6
A transaction processing executive under GCOS 6.

Trace Level
This is a number from 0 to 5 that indicates the level of detail to be included in the CPI-C traces. It has also been called ”Logging Level”.

Trace Storage Mode
The mode determines how the trace information is stored. It can be either FILE or BUFFER. This is also known as ”Logging Flag”.

TS8
A GCOS 8 module.

TSS
Time Sharing Software is part of GCOS 8.

X/Open
The organization that develops international standards for UNIX.

Index

Numbers

2LTP, 1-4

A

A Symbol, 2-11
AD Identifier, A-5
AE Symbol, 2-11
Alphanumeric String, A-1
API, 1-1
API_LOG_FLG Variable, 3-2, 3-6, 3-14
API_LOG_LEV Variable, 3-1, 3-6, 3-14
API_MODE Variable, 3-1, 3-6, 3-13
api_msg Function, 1-3, 3-3
API_SSDUE Variable, 3-2
API_SSDUR Variable, 3-2
API_TERM_NAME Variable, 3-7
API_TP_NAME Variable, 3-6
AUT_LOG_FILE Variable, 2-3, 3-2, 3-13

B

Break Function, 3-27
buffer Parameter, 3-17, 3-24

C

CCPSTID.o Object, 2-12
CICS, 1-7
CMACCP, 1-8, 2-1, 2-7, 2-8, 2-10, 2-16, 3-2, 3-6, 3-20
CMALLC, 1-3, 2-1, 2-5, 2-7, 2-8, 2-10, 3-9, 3-13, 3-15
CMDEAL, 1-3, 2-1, 2-8, 2-10, 2-11, 3-11
CMINIT, 1-3, 1-8, 2-1, 2-5, 2-7, 2-8, 2-10, 2-16, 3-13, 3-20
CMRCV, 1-3, 2-1, 2-6, 2-7, 2-8, 2-10, 2-11, 3-16
CMSDT, 1-3, 3-20
CMSED, 1-3, 2-1, 3-22
CMSEND, 1-3, 2-1, 2-5, 2-6, 2-8, 2-10, 3-18, 3-24
CMSERR, 1-3, 2-1, 2-10, 3-23, 3-26
CMSTD, 2-1
cob Command, 2-12
COB_errmsg Variable, 3-3
COBOL, 2-12, 3-4
COBOL 85, 2-1
COBOL II, 2-1
COBOL/2, 2-12
cobrun, 2-12
Compiler, 2-12
Configuration File, 2-14, 3-14
Configurations, 1-5
Connection Number, A-1
Conversation ID (example), 2-16
Conversation ID (pseudo), 2-15
CPI-C SS Verbs, 3-1
CPI-C Library, 1-1
cpic_startup Command, 2-3
cpic-c.h Header File, 3-3
cpi-clog Command, 3-2, A-5
CXI, 1-6

D

DA Symbol, 2-11
DAC, 1-6
Data Flow, 3-23
data received Parameter, 3-17
Datanet, 1-5, 1-6, 1-7, 3-7
Deallocate Calls, 3-27
deallocate type, 3-11
Debugging, 2-1, 2-14
Diagnostic Functions, 1-3
DMIVTP, 1-6
DMVITP, 1-5
DN Symbol, 2-11
DPS 7000, 1-5, 2-9
DPS 9000, 1-6
DPX, 2-5
DPX/DPX Connections, 3-22
DR Symbol, 2-11
DST8, 2-9
DTF, 1-5
dumpcpic, 2-14, 2-16, A-5

E

ECL, 1-5
elgfile, 3-2, A-5
ELOG Driver, 3-2, A-5
elog_print File, 3-2, A-5
EMU_MODEL Variable, 3-14
EN Symbol, 2-11
Environment Variables, 3-1, A-6
EP Symbol, 2-11
ernum, A-5
errmsg Output Area, 3-3
Error Direction, 3-22
Error Messages, A-1
Export File, 2-13

G

Gateway, 1-6
GCOS, 1-1, 2-5, 3-7, 3-11, 3-22, 3-27
GCOS 6, 1-4, 1-5
GCOS 7, 1-4, 1-5
GCOS 7 TP, 1-5
GCOS 8, 1-4, 2-5
GCOS6 TP, 1-5
gnt Files, 2-12

H

Hexadecimal Code, A-1

I

IBM, 1-4, 1-7, 2-5
Include Directory (private), 2-13
inftms, 2-9
INIT, 2-10

INITIAL_COR Variable, 3-14
IOF, 1-5
ISO Sessions, 1-1

K

keymsg Error Message Code, 3-3

L

lgprint Command, 3-2, A-5
libmef.a Library, 2-13
libtmpi.a Library, 2-13
Linker, 2-12
LLC, 1-8
Local
  Mailbox, 3-6
  Program, 3-26
  System, 1-1
login, 2-3
LOGNAME Variable, 3-2, 3-6, 3-8

M

Mapped Conversation, 3-9
MDNET, 1-6
Microfocus, 2-1, 2-12
MROUT, 1-6
myfile Pointer, 3-3
myprog.cbl Program, 2-13

N

Native Code Generator, 2-12
ND Symbol, 2-11
NO Symbol, 2-11
Non Transparent Mode, 2-3

O

Object File Format (OFF), 2-12
OSF, 1-7
OSI Stack, 1-4, 1-5, 1-7, 1-8
OTM, 1-4
outfile, A-5

P

PC Symbol, 2-11
PE Symbol, 2-11
PID, 1-5, 1-6
pmaderror retcode, A-1

R

R Command, 2-4
rdmessg Function, 3-1, 3-3
Receive Data, 2-3
RECEIVE Return Code, 3-10
RECEIVE State, 2-7
receive type Parameter, 3-19
received length Parameter, 3-17
RECV, 2-10
REMMB_EXT Variable, 3-14
Remote
  Application, 1-4
  Machine, 1-4
  Program, 3-23, 3-26
  Site, 2-8
  System, 1-1
request_to_send Notification, 3-27
request_to_send_received Parameter, 3-17, 3-25, 3-26, 3-27
requested length Parameter, 3-17
RESET, 2-10
retrieve_error Function, 1-3, 3-1, 3-5
Return Codes, A-4
return control Parameter, 3-9
RF Symbol, 2-11
RS Symbol, 2-11
rts32, 2-12
Run Time System (RTS), 2-12

S

S Command, 2-3
scancpic, 2-14
SE Symbol, 2-11
SECURITY Variable, 3-14
SEND, 2-10
Send Data, 2-3, 3-23, 3-27
SEND Return Code, 3-10
SEND State, 2-7, 2-11
send type Variable, 2-7
send_length Parameter, 3-25
Service Provider, 1-4
Session Connection, 3-8
Session Layer, 1-4, 1-8
Session User, 2-8
SMIT, 2-8, 2-14, A-6
SSDU, 1-1, 3-2
State Transition Tables, 2-10
status received Parameter, 3-17
Symbolic Destination Name, 2-9, 2-16

T

TDS, 1-5
Terminal Management, 1-1
tmcall, 2-9
tmp/api.trc File, 2-14, A-5
tmp/api.trc file, 1-8
tmp/apilog File, 2-15, 3-2
tmp/aplilog File, A-5
tmp/dumpcpic File, 2-16
Token, 1-3, 1-4
TP, 1-5
TP Form, 2-5, 2-6
TP8, 1-6
TPS6, 1-5
Trace Facility, 2-1, 2-14, A-5
Trace File, 2-15
Trace Level, 2-14, A-5
Trace Storage Mode, 2-14, A-6
Traces (OSI User), 2-14
Transparent Mode, 1-1, 2-2, 3-10
Transport Layer, 1-8
TS8, 1-6
TSS, 1-6
Two Level Transaction Processing, 1-4

U

User Application, 1-1, 1-5
User Transactions, 1-5
USERINFO, 3-13
usr/cpic Directory, 3-3
usr/cpi-c/site.cnf Flie, 3-14
utmlog Command, A-5

V

Variables, 3-1
VIP Protocol, 1-1
VTAM, 1-7

W

W Symbol, 2-11

X

X/OPEN, 1-3
X25.3 Mapper, 1-8

Vos remarques sur ce document / Technical publication remark form

Titre / Title :	Bull DPX/20 Open Terminal Management (OTM) CPI-C SS Bull Environment User's Guide

Nº Reférence / Reference Nº : 86 A2 32PE 04	**Daté / Dated :** April 1996

ERREURS DETECTEES / ERRORS IN PUBLICATION

AMELIORATIONS SUGGEREES / SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Vos remarques et suggestions seront examinées attentivement
Si vous désirez une réponse écrite, veuillez indiquer ci-après votre adresse postale complète.

Your comments will be promptly investigated by qualified technical personnel and action will be taken as required.
If you require a written reply, please furnish your complete mailing address below.

NOM / NAME : ______________________________ Date : ________

SOCIETE / COMPANY : ______________________________

ADRESSE / ADDRESS : ______________________________

____

Remettez cet imprimé à un responsable BULL ou envoyez-le directement à :

Please give this technical publication remark form to your BULL representative or mail to:

Bull Electronics Angers S.A.
CEDOC
Atelier de Reprographie
331 Avenue Patton
49004 ANGERS CEDEX 01
FRANCE

PLACE BAR CODE IN LOWER
LEFT CORNER

Bull Electronics Angers S.A.
CEDOC
Atelier de Reprographie
331 Avenue Patton
49004 ANGERS CEDEX 01
FRANCE

ORDER REFERENCE
86 A2 32PE 04

Bull

Utiliser les marques de découpe pour obtenir les étiquettes.
Use the cut marks to get the labels.

DPX/20
AIX
OTM
CPI-C SS Bull
Environment User’s
Guide
86 A2 32PE 04

DPX/20
AIX
OTM
CPI-C SS Bull
Environment User’s
Guide
86 A2 32PE 04

DPX/20
AIX
OTM
CPI-C SS Bull
Environment User’s
Guide
86 A2 32PE 04