Connect:Express®
UNIX

Integration Tools Guide

Version 1.4.6

Sterling Commerce
An IBM Company

Connect:Express UNIX Integration Tools Guide
Version 1.4.6
First Edition

Contents

Preface

Chapter 1 Application Programming Interface (API)

About the API 1-1

Implementing the API 1-2
ZREQ_TOM Structure 1-3
Application Program Example 1-4

Submitting a Transfer Request (Example) 1-4
Call Parameters 1-4
Return Codes 1-7

Chapter 2 Utilities

About the Utilities 2-1
Call Parameters 2-2
Return Codes 2-3

Implementing the Utilities (Examples) 2-5
Requesting a Transfer 2-5
Interrupting a Request 2-6
Deleting One or More Requests 2-7
Deleting a Partner 2-7
The End to End utility 2-8
Acknowledging a Transfer 2-8
Forwarding a Transfer 2-8
P1b8pe2e Reference 2-8

Chapter 3 Exits and Procedures

About User Exits 3-1
Implementing User Exits 3-3
Processing with a User Exit (Example) 3-4

About User Procedures 3-4
Implementing User Procedures 3-5
Parameters 3-6
Processing with an Integrated User Procedure (Example) 3-8

Chapter 4 The TOM_PRM Command

About the TOM_PRM Command 4-1
Syntax of tom_prm 4-2

Using the TOM_PRM Command 4-2
Extracting to a Text File 4-2
Extracting to a Binary File 4-2
Reinitializing the Database 4-3
Uploading Data to the Database from a Text File 4-3
Uploading Data to the Database from a Binary File 4-3
Modifying an Element in the Database 4-3
User out and Error out 4-4

Text Files Containing Directives 4-4
DATABASE Directive 4-4
PARTNER Directive 4-5
FILE Directive 4-6
SESSION Directive 4-7
PRESENTATION Directive 4-7
EXTRACT Directive 4-8

Binary Extract Files 4-8

Extracting all Parameters from the Database 4-11

Appendix A Structure Files

d0b8z20.h File A-1

d1b8ruex.h File A-7

Appendix B Compilation Procedures

Compilation Procedure B-1

Make Executable Procedure B-2

Preface

The *Connect:Express UNIX Integration Tools Guide* is for users and system administrators of Connect:Express UNIX.

This guide assumes some knowledge of the UNIX operating system, including its applications, network, and environment. If you are not familiar with the UNIX operating system, refer to the UNIX library of manuals.

About the Integration Tools

Connect:Express UNIX ships with the following integration tools to help you manage transfer operations.

Integration Tool	Description
Application Programming Interface (API)	The API can be implemented in C language and enables you to configure and implement file transfers from an application. The API uses a Message Queue to connect to the monitor and submit transfer requests to Connect:Express.
Utilities	The utilities are built on the API and enable you to define and manage transfers using shell scripts. The utilities are batch programs that can be called from a shell, and pass parameters using keywords.
User exits	The exit interface enables you to integrate user exits into the transfer execution process. User exits run synchronously with the transfer process, for example you can activate user exits at the beginning and end of reception or at the beginning and end of transmission. You use the symbolic file definition to specify which user exit to run and when to activate it.
Integrated user procedures	An integrated user procedure is launched by the monitor and executes asynchronously,outside of transfer operations. It is a shell procedure that enables you to activate a process during transfer operations, for example you can activate user procedures at the beginning and end of reception or at the beginning and end of transmission. The symbolic file definition enables you to integrate user procedures when a transfer executes.
ETEBAC3 exit	The ETEBAC3 exit enables you to manage the ETEBAC3 card in any format. In the server mode, the exit interprets the received card and processes it under the control of the application. A data set (symbolic partner name, symbolic password, symbolic file name) is returned to Connect:Express at the end of the exit for standard processing by the monitor. In client mode, you can build an ETEBAC3 card specifically for a Symbolic partner/Symbolic file pair. Refer to the Connect:Express UNIX ETEBAC3 Guide for more information.

Integration Tool	Description
Translation tables	External translation tables support the exchange of data between all platforms. For example, you may need to translate data from ASCII to EBCDIC before transmission or from EBCDIC to ASCII after reception. When working with many environments, you may need to use several translation tables. You can use the sample translation table to create new ones and then refer to them in the symbolic file definition.

The following table identifies the directories that contain files for these integration tools.

Directory	Contents
/itom	Programming interface and utilities Examples
/exit	User exits and procedures Generalized procedure UEXERR Examples Input parameters file for exits and user procedures Output trace files for exits and user procedures
/config	Translation tables
/notif	Notification utilities

Chapter Overview

The *Connect:Express UNIX Integration Tools Guide* is organized into the following chapters and appendices:

Chapter/Appendix	Description
Chapter 1 API	This chapter describes how to implement the Application Programming Interface (API) and provides an example of a transfer request.
Chapter 2 Utilities	This chapter provides an overview of the utilities and includes examples of calls.
Chapter 3 Exits and Procedures	This chapter provides an overview of exits and user commands for start and end of transfer, and describes how to implement them.
Chapter 4 The TOM_PRM Command	This chapter describes the TOM_PRM command.
Appendix A Structure Files	This appendix provides examples of the structure files, d0b8z20.h and d1b8ruex.h.
Appendix B Compilation Procedures	This appendix provides examples of the compilation procedure and the Make executable.

Connect:Express Documentation

Connect:Express documentation consists of the following manuals:

- The *Connect:Express UNIX User and Installation Guide* is for administrators that install and configure Connect:Express UNIX, and for users that execute file transfers. This document is only available in English.
- The *Connect:Express UNIX FTP Guide* provides you with the information that you need to use Connect:Express with the FTP protocol. This document is available in French and English.
- The *Connect:Express UNIX PeSIT User Fields Guide* describes how you can exchange the PeSIT Pi37 and Pi99 fields with any PeSIT software. This document is only available in French.
- The *Connect:ExpressUNIX Etebac3 User Guide* provides you with the information that you need to use Connect:Express with the Etebac3 protocol. This document is only available in French.

Getting Support for Sterling Commerce Products

Sterling Commerce provides intuitive technical products and superior Help and documentation to enable you to work independently. However, if you have a technical question regarding a Sterling Commerce product, use the Sterling Commerce Customer Support Web site.

The Sterling Commerce Customer Support Web site at *www.sterlingcommerce.com* is the doorway to Web support, information, and tools. This Web site contains several informative links, including a solutions database, an issue tracking system, fix information, documentation, workshop information, contact information, sunset and retirement schedules, and ordering information. Refer to the Customer Support Reference Guide at *www.sterlingcommerce.com/customer/tech_support.html* for specific information on getting support for Sterling Commerce products.

Chapter 1

Application Programming Interface (API)

This chapter describes how to implement the Application Programming Interface (API) and provides an example of a transfer request.

About the API

You can access administration functions for the monitor using the API, which is written in C language. You must complete certain fields in the ZREQ_TOM structure and then the API calls the L0B8Z20 function to communicate with the monitor. Parameters are passed to the monitor in the ZREQ_TOM structure. When the function is complete, the structure ZREQ_TOM returns the return codes of the call and the values if there was a failure.

You can use the API to send requests to the monitor to execute any of the following functions:

Function	Description
Managing transfers (RENC file)	You can submit a transfer request, interrupt a data transfer, delete one or more transfers based on criteria that you specify, restart a data transfer, or display data about a transfer.
Acknowledging transfers (RENC file)	You can submit a end to end transfer acknowldegment request (EERP), from request or providing the initial transfer parameters
Forwarding transfers (RENC file)	You can submit a forward transfer request , from request or providing the initial transfer parameters
Managing partners (RPAR file)	You can view, create, modify, and delete a symbolic partner definition. The following source programs are provided as examples: p1b8ppar_c and p1b8ppar_m.
Managing files (RFIC file)	You can view, create, modify, and delete a symbolic file definition. The following source programs are provided as examples: p1b8pfil_c and p1b8pfil_m.
Managing statistics (RENC file)	You can display results of a transfer request.

The following table describes the components provided in the itom and itom/SAMPLES directories.

Module	Description
libitom.a	Library containing the object LOB8Z20.o

Module	Description
d0b8z20.h	Communication structure between an application program and the Connect:Express interface. This structure contains descriptions of Connect:Express files, for example RPAR,RFIC and RENC.
SAMPLES	Directory of sample files
p1b8pcan.c	Source file c: example of interrupting a transfer
p1b8pfil_c.c	Source file c: example of creating a file
p1b8pfil_d.c	Source file c: example of viewing a file
p1b8pfil_m.c	Source file c: example of updating a file
p1b8pfil_s.c	Source file c: example of deleting a file
p1b8ppar_c.c	Source file c: example of creating a partner
p1b8ppar_d.c	Source file c: example of viewing a partner
p1b8ppar_m.c	Source file c: example of updating a partner
p1b8ppar_s.c	Source file c: example of deleting a partner

Implementing the API

To implement the API, you must include the following components in the source code of your user program:

- The structure ZREQ_TOM which is described in the file d0b8z20.h. Refer to Appendix A for details about this file.
- The option -L which points to the environment variable that corresponds to the root directory for the monitor, for example /home/tom1

```
-L $TOM_DIR/itom
```

- A call to the function L0B8Z20 which is included in the library *libitom.a*, located in the directory designated by the environment variable $TOM_DIR/itom. It must be link-edited with the calling program. See Appendix B for an example of the compilation script and the MAKE file of the user program.

The API uses message queues to communicate with the monitor (IPC system V). The user program creates a message queue inside the L0B8Z20 function. Then, it places its request in the monitor's message queue, so the identifier is TOM_DIR. This request contains the identifier of the message queue for the user program and enables the monitor to send the results of the request after processing. Before returning to the user program, the L0B8Z20 function deletes the message queue corresponding to the call. This use of message queues is transparent to the user, but it means that the monitor must be active for a user program to work.

ZREQ_TOM Structure

The structure ZREQ_TOM has two parts, a header and a structure. The header is the same for all APIs, but the structure depends on the call type to the API. The following table lists the four possible call types.

Structure	Description
structure st_sci	Submits a transfer request to the monitor.
structure s_renc	Accesses the requests file.
structure partenaire	Accesses the partners file.
structure fichier	Accesses the symbolic files file.

Header

```
struct ZREQ_TOM {
     char      zreq_tom_name[4];    /*   Monitor name           */
     char      zreq_tom_func[1];    /*   Function type          */
     char      zreq_tom_tabn[1];    /*   Call type to the API   */
     char      zreq_tom_reqn[8];    /*   Request number         */
     char      zreq_tom_rtcf[1];    /*   Monitor return code    */
     char      zreq_tom_rscf[3];    /*   Reason return code     */
     union uni_sci uni;
};
```

Call structure

```
union uni_sci {
     struct          st_sci zreq_tom_sci;
     struct          s_renc zreq_tom_renc;
     struct           partenaire zreq_tom_part;
     struct           fichier zreq_tom_fic;
};
```

The header displays the function type in the first code (Transfer, Modification, Deletion, etc.), and indicates the file accessed (Requests, Files, Partners) with a subcode. The following table provides a summary of available functions:

Function Type	File Accessed	Service
T = Transfer request	R = Requests File: RENC	TR
I = Transfer interrupted		IR
P = Purged transfer		PR
R = Restart an interrupted transfer		RR
D = Display information		DR
E = EERP		ER
F = Forward a transfer		FR
C = Create	P = Partners File: RPAR	CP
M = Modify		MP
S = Delete		SP
D = Display information		DP

Function Type	File Accessed	Service
C = Create	F = Files File: RFIC	CF
M = Modify		MF
S = Delete		SF
D = Display information		DF

Application Program Example

The following example shows a simple application program.

```
#include <d0b8z20.h>
ZREQ_TOM *zreq;
..................
zreq = malloc(sizeof(ZREQ_TOM));
/* Loading values in the zreq structure zreq for the function that you want */
..................
ret = L0B8Z20(zreq).
/* Test of return codes in the structure zreq */
..................
free(zreq);
..................
exit(0);
```

Submitting a Transfer Request (Example)

Using the API, you can submit a request, interrupt and delete a request, restart an interrupted transfer, and display information about a transfer. When a request is submitted, the interface receives a request number. This request number is used later with other functions for this request. The following example shows how you can use the API to submit a transfer request:

```
int L0B8Z20(struct ZREQ_TOM *);
```

This function enables you to place a transfer request in the monitor's message queue and to specify different characteristics such as direction, symbolic file name, or partner name.

Call Parameters

L0B8Z20 is called with a ZREQ_TOM structure in a parameter. The length of each field is indicated in brackets. For example, zreq_tom_name[4] has a length of 4 characters.

The parameters that are not marked required are optional when default values exist for these fields. Default values come from the symbolic file definition for the transferred file. If these values are not provided in the call, they must be initialized, except when indicated with a space.

The parameters are identical to those used by the STERM utility to execute a transfer request. Refer to STERM documentation for more information about these parameters.

The following table describes the parameters of the header.

Field	Description	Note
zreq_tom_name[4]	Monitor name, usually TOM1	Required
zreq_tom_func[1]	Function code of the API «T» Transfer service	Required
zreq_tom_tabn[n]	Subfunction code «R» Requests file (RENC)	Required
zreq_tom_reqn[8]:	Request number. Must be entered as a zero binary (x'0') to the call.	Response
zreq_tom_rtcf[1]:	Monitor return code. Must be entered as a zero binary (x'0') to the call.	Response
zreq_tom_rscf[3]	Supplementary reason code. Must be entered as a zero binary (x'0') to the call.	Response

The following table describes the parameters for the part that is specific for the call type to the API:

Field	Description	Note
uni.zreq_tom_sci.dire[1]	Transfer direction. Valid values: T - Transmission R - Reception	If not entered (spaces), the value from the profile file is used.
uni.zreq_tom_sci.file[8]	Name of profile file (Symbolic file name). 1 to 8 alphanumeric characters. Completed with spaces	Required.
uni.zreq_tom_sci.part[8]	Symbolic partner name. 1 to 8 alphanumeric characters. Completed with spaces	If not entered (spaces), the value from the profile file is used.
uni.zreq_tom_sci.dsnam[44]	Physical file name. 1 to 44 alphanumeric characters. Completed with spaces	If not entered (spaces), the value from the profile file is used.
uni.zreq_tom_sci.prty[1]	Transfer priority. Valid values: 0 - (x'30') 1 - (X'31') 2 - (X'32').	If not entered (spaces), the value from the profile file is used.
uni.zreq_tom_sci.dat[8]	Date requested for the start of transfer in the format YYYYMMDD.	If both the date and time are not entered, the request is executed immediately.
uni.zreq_tom_sci.hour[6]	Time requested for the start of transfer in the format HHMMSS.	If both the date and time are not entered, the request is executed immediately.
uni.zreq_tom_sci.lnk[1]	Type of network interface used. Valid values: T - TCP/IP X - X25 P - PAD	If not entered, the value of the partner is used.
uni.zreq_tom_sci.udf[44]	User data.	

Field	Description	Note
uni.zreq_tom_sci.typ[1]	Request type. Valid values: N - Normal I - Inquiry: Enables you to receive a file from a remote partner using a request type. Only for transmitting mode. H - Hold: Making a file available for transfer. The request will be selected by the remote partner using a request type. Only for receiving mode.	
uni.zreq_tom_sci.sta[1]	Not used	Status of transfer
uni.zreq_tom_sci.dpcsid[8]	DPCSID Alias. Name of the local monitor that calls the remote monitor.	If not entered (spaces), the value from the partner is used. If the value for the partner is not entered, the value from the DPCSID of the sysin file is used.
uni.zreq_tom_sci.dpcpsw[8]	DPCPSW Alias. Password of the local monitor that calls the remote monitor.	If not entered (spaces), the value from the partner is used. If the value for the partner is not entered, the value from the DPCSID of the sysin file is used.
uni.zreq_tom_sci.format[2]:	Format of the registered file. TF - Text file with records of a fixed length. TV -Text file with records of variable length. BF - Binary format with records of a fixed length. BU - Binary format undefined.	If not entered (spaces), the value from the profile file variable is used.
uni.zreq_tom_sci.lrecl[5]	Records with a length of 5 numeric characters.	If not entered (spaces), the value from the profile file is used.
uni.zreq_tom_sci.api[88]	API field. In Etebac3, this field corresponds to the Etebac3 card.	
uni.zreq_tom_sci.tsm[3]	Type, structure, and mode for FTP transfers. Valid values: A - Ascii E - Ebcdic B - Binary * - Unchanged Structure that can have the following values: «F» File, «R» Record, «*» Unchanged mode that can have the following values: «B» Block, «S» Stream, «*» Unchanged	If not entered (spaces), the value from the profile file is used.
uni.zreq_tom_sci.stou[1]	Indicator store unique for FTP transfers. Valid values: Y - Yes N - No	If not entered (spaces), the value from the profile file is used.
uni.zreq_tom_sci.fa[1]	Indicator to use the file in reception by the Connect:Enterprise File Agent. Valid values: Y - Yes N - No	If not entered (spaces), the value from the profile file is used.
uni.zreq_tom_sci.label[80]	PeSIT Pi37	In transmission only.

Field	Description	Note
uni.zreq_tom_sci.s_pi99_254[254]	PeSIT Pi99	With a partner of type Other.
uni.zreq_tom_sci.user_org[8]	Origin of the transfer	Pi3 bis
uni.zreq_tom_sci.user_dst[8]	Destination of the transfer	Pi4 bis
uni.zreq_tom_sci.user_snd[24]	Sender of the file	Pi61
uni.zreq_tom_sci.user_rcv[24]	Receiver of the file	Pi62
uni.zreq_tom_sci.quant_aa[2]	Year AA for the Julian date	
uni.zreq_tom_sci.quant[3]	Julian date	
uni.zreq_tom_sci.notif[1]	Notification flag	

Return Codes

You can test the value of the return code of the call to L0B8Z20 as follows:

```
status = L0B8Z20(param);
```

If status = 0, there was no error. If status = 2, there was an error. The structure ZREQ_TOM contains the results of the request.

Field	Description
zreq_tom_reqn[8]:	Contains the request number assigned by the transfer monitor.
zreq_tom_rtcf[1]:	If zreq_tom_rtcf[1] = 0 no error, if zreq_tom_rtcf[1] <> 0 error.
zreq_tom_rscf[3]	Contains the TRC (Error code returned to the API by the monitor).

The following screen shows an example.

```
#include "d0b8z20.h"
------
struct ZREQ_TOM *param;
#define SIZE_ZREQ
int L0B8Z20(struct ZREQ_TOM *);
int status = 0 ;
void main(int argc,char *argv[])
{
/* Communication structure  */
param = (struct ZREQ_TOM *)malloc(sizeof(struct ZREQ_TOM));
if (param==(struct ZREQ_TOM *)0) exit(1) ;
memset((char *)param, ' ', sizeof(struct ZREQ_TOM));
/* Initialization of the structure with the request parameters */
/* Header */
param->zreq_tom_func[0] = 'T';
param->zreq_tom_tabn[0] = 'R';
memset(param->zreq_tom_reqn, 0x00, 8);
memset(param->zreq_tom_rtcf, 0x00, 4);
/* Specific part */
param->uni.zreq_tom_sci.dire[0] = ‘T’ ;
memcpy(param->uni.zreq_tom_sci.file, "FILE01",6);
memcpy(param->uni.zreq_tom_sci.part, "PART01",6);
memcpy(param->uni.zreq_tom_sci.dsnam, "/home/tom1/out/f01.txt",22);
memcpy(param->uni.zreq_tom_sci.lnk[0] = ‘T’ ;

status = L0B8Z20(param);
if (status != 0) {
  /* Request not OK */
  free(param) ;
  fprintf(stderr, "%.4s\n", param->zreq_tom_rtcf);
  fflush(stderr);
  exit(1);
}

printf("%.8s\n",param->zreq_tom_reqn);
free(param) ;
exit(0);
}
```

Chapter 2

Utilities

This chapter provides an overview of the utilities and includes examples of calls.

About the Utilities

The utilities are batch programs built on the Application Programming Interface (API) described in chapter 1. These batch programs are written in C language and provide examples of how you can use the API. The programs pass parameters to the monitor using keywords, according to the following syntax:

```
p1b8preq "/SFN='symbolic file'/PRT=1/LNK=T/SPN='symbolic partner' "
```

The following table describes the available functions using these utilities.

Function	Description
Managing directories	You can modify or view symbolic Partners and Files directories from a shell. You can also delete and view a partner or file definition. The add and modify functions are provided in program sources. Use the following programs to delete and view a partner or file: p1b8ppar_s, p1b8ppar_d, p1b8pfil_s, p1b8pfil_d
Managing transfers	A shell can submit a transfer request, interrupt a request, delete it, or restart it using the following programs: p1b8preq, p1b8pcan, p1b8ppur, p1b8pret, p1b8pe2e.
Viewing statistics	A shell can request transfer results and statistics using the p1b8pren program.

These programs are delivered in the /itom directory. The following table describes the utilities in the /itom directory and identifies the Connect:Express file that is affected. RENC is the requests file, RPAR is the partners file, and RFIC is the file definitions file.

Module	Description	Connect:Express File
p1b8pcan	Interrupts a transfer request.	RENC
p1b8pe2e	Acknowledges or forwards a transfer	RENC
p1b8pfil_d	Displays a file definition from the symbolic files file.	RFIC

Module	Description	Connect:Express File
p1b8pfil_s	Deletes a file from the symbolic files file.	RFIC
p1b8ppar_d	Displays a partner definition from the partners file.	RPAR
p1b8ppar_s	Deletes a partner from the partners file.	RPAR
p1b8ppur	Deletes a transfer request.	RENC
p1b8pren	Sends a request to display transfer statistics.	RENC
p1b8preq	Submits a transfer request.	RENC
p1b8pret	Sends a request to restart a transfer.	RENC

Call Parameters

The following table describes the keywords used by the utilities. The last column indicates which Connect:Express file is affected. RENC is the requests file, RPAR is the partners file, and RFIC is the file definitions file.

Keyword	Value	Description	Connect:Express File
/REQ	8 digit number	Request number. Example: 10400023	RENC
/DAT	YYYYMMDDHHMMSS	Date and time entered in full or partially. Example: 20030801, 2003080112	RENC
/SFN	8 alphanumeric characters	Symbolic file name. Example: RAPPORT	RENCRFIC
/SPN	8 alphanumeric characters	Symbolic partner name. Example: BORDEAUX	RENCRPAR
/DIR	T, R,*	Transfer direction: Transmission, Reception, both	RENC
/PRT	0, 1, 2	Priority	RENC
/LNK	T, X, M	Type of network link: X25, TCP/IP or Mixed	RENC
/DSN	1 to 44 alphanumeric characters	Physical name of the local UNIX file	RENC
/UDF	1 to 44 alphanumeric characters	User data. For example, physical file name on the remote server	RENC
/MNM	4 characters	Monitor name. Example: TOM1	RENC
/SID	1 to 8 alphanumeric characters	Local symbolic name (Alias)	RENC
/PSW	1 to 8 alphanumeric characters	Local password (Alias)	RENC
/RFM	TF, BF, TV, BU, T*, B*	Record format for records: Text, Binary, Fixed, Variable, Undefined …	RENC

Keyword	Value	Description	Connect:Express File
/RLG	1 to 5 alphanumeric characters	Record length	RENC
/API	1 to 88 alphanumeric characters	ETEBAC3 card	RENC
/TYP	N, H, I	Request type: Normal, Hold, Inquiry	RENC
/TSM	A/E/B, F/R, B/S	FTP. Data type: File structure, Transfer mode	RENC
/STO	Y/N	FTP. Option Store Unique: yes/no	RENC
/LAB	1 to 80 alphanumeric characters	PeSIT: Pi37, label	RENC
/P99	1 to 254 alphanumeric characters	PeSIT: Pi99 only for partner of type Other	RENC
/ORG	1 to 8 alphanumeric characters	Origin of the transfer	RENC
/DST	1 to 8 alphanumeric characters	Destination of the transfer	RENC
/P61	1 to 24 alphanumeric characters	Sender of the file	RENC
/P62	1 to 24 alphanumeric characters	Receiver of the file	RENC
/QQQ	3 digit number	Julian Date	RENC
/NTF	0-7	Notification flag	RENC
/P11	4 hexadecimal characters	File type (pi11)	RENC
/P12	1 to 14 alphanumeric characters	File name (pi12)	RENC
/P13	1 to 8 numeric characters	Transfer identification (pi13)	RENC
/P51	1 to 12 numeric characters	File date (pi51)	RENC

Return Codes

All of the utilities return the same codes in the $? variable of the shell. The following table describes the codes returned by the utilities.

Return code	Description
0	Function executed successfully.
1	Number of arguments is incorrect.
2	Error detected by the utility: see the supplementary code (Next table).
3	Error detected by Connect:Express. Refer to Appendix B Return Codes in the Connect:Express UNIX User Guide for a list of codes.

The following list identifies the supplementary codes associated with the return code 2. These codes have the format XYYZ, where X = argument number of the call, YY = field in error (SFN, SPN, etc.), and Z = error type (invalid parameter, repeated parameter, etc.).

```
/* Internal Error Return Code */
#define ERROR_BAD_FUNC 2900
#define ERROR_CRE_QUEUE 2901
#define ERROR_PB_SEND 2902
#define ERROR_PB_RECV 2903
#define ERROR_TIME_OUT 2904
#define ERROR_NOTOM 2912
#define ERROR_OTHER 2999

/* External Error Status (4 digits) : XYYZ */
/* X : argument number (1,2,3) */
/* YY : Field which contains error */
/* Z : Error type */

/* YY */
#define Y_OTH 0  /* Other */
#define Y_PRT 1  /* Priority */
#define Y_DIR 2  /* Direction */
#define Y_LNK 3  /* Link */
#define Y_SPN 4  /* Partner */
#define Y_SFN 5  /* File */
#define Y_DSN 6  /* Physical Name */
#define Y_UDF 7  /* User Data Field */
#define Y_DAT 8  /* Date */
#define Y_MNM 9  /* Monitor */
#define Y_REQ 10 /* Request Number */
#define Y_SID 11 /* Alias Name */
#define Y_PSW 12 /* Alias Password */
#define Y_RFM 13 /* Record Format */
#define Y_RLG 14 /* Record Length */
#define Y_API 15 /* Api */
#define Y_STA 16 /* State */
#define Y_TYP 17 /* Request Type */
#define Y_TSM 18 /* Type/Struct/Mode FTP */
#define Y_STO 19 /* Store/Unique FTP */
#define Y_FAG 20 /* File agent flag Y/N */
#define Y_LAB 21 /* Label */
#define Y_P99 22 /* PI99 254 char */
#define Y_ORG 23 /* User Origin */
#define Y_DST 24 /* User Destination */
#define Y_P61 25 /* PI61 */
#define Y_P62 26 /* PI62 */
#define Y_QQQ 27 /* Julian date */
#define Y_NTF 28 /* Notification flag */
#define Y_P11 29 /* Eerp/snf pi11    */
#define Y_P12 30 /* Eerp/snf pi12    */
#define Y_P13 31 /* Eerp/snf pi13    */
#define Y_P51 32 /* Eerp/snf pi51    */
#define Y_ACK 33 /* Eerp ACK    */
#define Y_FUN 34 /* Eerp or FWD    */

/* Z */
#define Z_INV_FIELD 1 /* Invalid Field */
#define Z_DUP_FIELD 2 /* Duplicate Field */
#define Z_LG_FIELD 3  /* Invalid Field Length */
#define Z_MIS_FIELD 4 /* Missing Compulsory Field */
```

Implementing the Utilities (Examples)

The following screen shows an example of a simple procedure:

```
#
#   Sample to Request a Transfer to Monitor.
#
$TOM_DIR/itom/p1b8preq "/SFN=FILE/SPN=PART1/DIR=T" > reqnumb.txt
code=$?
echo Return Code $code
cat reqnumb.txt
```

In this example, the request number is in the file *reqnumb.txt*, and the return code is in the $? variable. If the transfer was accepted by Connect:Express, the file *reqnumb.txt* contains a value.

This section provides specific examples of how you can use the utilities. Processing for a partner and for a file are similar.

Requesting a Transfer

From a shell procedure, you can submit a transfer request to transfer the file FILE with priority 1, using a TCP/IP link with the partner PART. The name of the file to transfer is TOM.tmp, and the text string “user information” is sent in the Pi99 field.

```
$TOM_DIR/itom/p1b8preq "/SFN=FILE/PRT=1/LNK=T/SPN=PART" "/DSN=/tmp/TOM.tmp"
                       "/P99=user information"
```

Request parameters are made up of four arguments. The first argument represents the transfer definition. Arguments 2, 3 and 4 are taken from 4 possibilities: Physical file name, label, user field Pi99 (/UDF if type Tom or /P99 if type Other, or the API field.

Argument	Field	Length	Comments
Transfer Definition (Arg 1)	Symbolic file name (/SFN=...)	1 to 8 alphanumeric characters	Required
	Symbolic partner name (/SPN=...)	1 to 8 alphanumeric characters	Default in the RFIC definition
	Priority (/PRT=...)	0, 1, or 2	Default in the RFIC definition
	Link type (/LNK=...)	T, P or X	Default in the RPAR definition
	Date and time (/DAT=...)	yyyymmddhhmmss	Default is the current date/time.
	Direction (/DIR=...)	T or R	Default in the RFIC definition
	Request type (/TYP=...)	N, I or H	Default N
	Dpcsid Alias (/SID=...)	1 to 8 alphanumeric characters	Default in the RPAR definition
	Dpcpsw Alias (/PSW=...)	1 to 8 alphanumeric characters	Default in the RPAR definition
	Origin (/ORG=…)	1 to 8 alphanumeric characters	Optional
	Destination(/DST=…)	1 to 8 alphanumeric characters	Optional
	Sender (/P61=...)	24 alphanumeric characters	Optional
	Receiver (/P62=...)	24 alphanumeric character	Optional
	Record format (/RFM=...)	2 alphanumeric characters (TV, TF, BU, BF)	Default in the RFIC definition
	Record length (/RLG=...)	5 alphanumeric characters	Default in the RFIC definition
	FTP format (/TSM=...)	3 alphanumeric characters A,E,B,* F,S,* B,R,*	Default in the RFIC definition
	FTP Store Unique (/STO=...)	Y,N	Default in the RFIC definition
	Notification (/NTF=...)	0, 1, 2, 3, 4, 5, 6, 7	Default in the RFIC definition
Physical Name (Arg 2,3 or 4)	Physical file name (/DSN=...)	1 to 44 alphanumeric characters	Default in the RFIC definition
User Data Definition (Arg 2,3 or 4)	User data (/UDF=...)	1 to 44 alphanumeric characters	Optional
Label Definition (Arg 2,3 or 4)	Label (/LAB=...)	1 to 80 alphanumeric characters	Optional
P99 Field (Arg 2,3 or 4) Note: Only valid with partners of type Other.	Pi99 (/P99=...)	1 to 254 alphanumeric characters	Optional PeSIT free field
API Field (Arg 2,3 or 4)	API field(/API=...)	1 to 88 alphanumeric characters	Optional

Interrupting a Request

From a shell procedure, you can interrupt a transfer, by passing the transfer request number in an argument to the utility P1B8PCAN, as shown in the following example:

```
$TOM_DIR/itom/p1b8pcan /REQ=10400065
```

Deleting One or More Requests

From a shell procedure, you can delete a single request by request number, delete a group of request using criteria, or delete all requests. The utility P1B8PPUR processes the following arguments:

Argument	Description
To delete one request: /REQ=QQQNNNNN	Request number to delete
To delete several requests using a filter: /DAT=YYYYMMDDHHMMSS /DIR= (R,T or *) /SFN= /SPN= /QQQ=AAqqq /STA=(A,C,D,E,H,J,K or O)	 Deletes requests made prior to this date and time. Deletes requests in the transfer direction that you specify. Deletes requests for a symbolic file name. Deletes requests for a partner Deletes requests made prior to this Julian date Deletes requests with a specific status
To delete all requests: No arguments are needed	

The following table shows some examples.

Command	Description
$TOM_DIR/itom/p1b8ppur /REQ=10400065	Deletes the request number 10400065.
$TOM_DIR/itom/p1b8ppur /DAT=20030801	Deletes all requests made before August 2003.
$TOM_DIR/itom/p1b8ppur /QQQ=06020	Deletes all requests made before January 20, 2006.
$TOM_DIR/itom/p1b8ppur /DAT=20030801 /SFN=FIC	Deletes all requests for the symbolic file FIC made before August 2003.
$TOM_DIR/itom/p1b8ppur /STA=E	Deletes all requests with status E (ended transfers)
$TOM_DIR/itom/p1b8ppur	Deletes all transfer requests in the RENC file.

Deleting a Partner

From a shell procedure, you can delete a partner using the symbolic name, as shown in the following example.

```
$TOM_DIR/itom/p1b8ppar_s /SPN=PARTNER
```

The End to End utility

The end to end utility, called p1b8pe2e, enables you to forward and acknowledge transfers of files and messages.

Acknowledging a Transfer

If the request is present in the RENC file, it is possible to acknowledge it by referencing its number, as shown below:

```
$TOM_DIR/itom/p1b8pe2e "/FUN=E/REQ=10400065/SPN=adjacent" "/ACK='feedback message'"
```

The SPN parameter is necessary if the initial node is not the adjacent partner.

If the request is no longer in the RENC file, all parameters from the initial transfer must be provided:

```
$TOM_DIR/itom/p1b8pe2e"/FUN=E/SPN=adjacent"
                      "/P12=filef/P11=XX/P03=oo/P04=dd/P13=id/p51=dh/p61=cc/p62=bb"
                      "/ACK='feedback message'"
```

Forwarding a Transfer

If the request is present in the RENC file, it is possible to forward it by referencing its number, as shown below:

```
$TOM_DIR/itom/p1b8pe2e "/FUN=F/REQ=10400065/SPN=adjacent"
```

The SPN parameter is required.

If the request is no longer in the RENC file, all parameters from the initial transfer must be provided:

```
$TOM_DIR/itom/p1b8pe2e"/FUN=F/TYP=N/SPN=adjacent"
                      "/P12=filef/P11=XX/P03=oo/P04=dd/P13=id/p51=dh/p61=cc/p62=bb"
```

P1b8pe2e Reference

This section provides the syntax rules and all parameters that apply to p1b8pe2e utility.

P1b8pe2e utility can receive one to five parameters, depending on the type of function used and the way the transfer definition is passed. Parameter #1 can provide general transfer request parameters such as priority, notification options, link , scheduling date etc ….

The tables below list the parameters and sub-parameters and provide a description and rules for each.

EERP - Request

This request refers to the reception initial request, using the /REQ= subparameter.

Argument	Field	Description	Required or default
#1	FUN	Function - E=EERP	Required
	REQ	Request number, 8 alphanumeric characters. Example: /REQ=09800005	Required
	SPN	Remote partner name (adjacent)	Required
	SID	Local name (alias)	RPAR/Sysin
	PSW	Local password (alias)	RPAR/Sysin
	NTF	Notification option	RFIC
	PRT	Priority	RFIC
	LNK	Link type	RPAR
	DAT	Scheduling date	Immediat
	FAG	File agent option	N
#2		Eerp acknowledgment (message or file) default from the $$EERP$$ definition.	RFIC/$$EERP$$
ACK		Eerp acknowledgment (message)	
DSN		Eerp acknowledgment (file)	

EERP - Transfer Definition

This request provides the initial request information. No /REQ= parameter is provided , all transfer information is provided in parameter #2.

Argument	Field	Description	Required or default
#1			
	FUN	Function - E=EERP	Required
	SPN	Remote partner name (adjacent)	Required
	SID	Local name (alias)	RPAR/Sysin
	PSW	Local password (alias)	RPAR/Sysin
	NTF	Notification option	RFIC
	PRT	Priority	RFIC
	LNK	Link type	RPAR
	DAT	Scheduling date	Immediat
	FAG	File agent option	N
#2		Transfer definition	Required
	ORG	Origine of transfer. 1 to 8 alphanumeric characters. (pi3)Example: /ORG=Orgtrf01	Required
	DST	Destination of transfer. 1 to 8 alphanumeric characters. (pi4)Example: /DST=DSTtrf01	Required
	P11	File type. 4 hexadecimal characters. (Pi11) Example: 01FA	Required
	P12	File name. 1 to 8 alphanumeric characters. (pi12) – RFIC definition. Example: /P12=Ftest01	Required
	P13	Transfer identification. 1 to 8 numeric characters. (pi13) Example /P13=18	Required
	P51	File creation date: 12 numeric characters. Example: /P51=040110092503	Required
	P61	Transfer sender: 0 to 24 characters. (pi61) Example: /P61=Client name	Required
	P62	Transfer receiver: 0 to 24 characters. (pi62) Example: /P62=Service name	Required
#3 ACK DSN		Eerp acknowledgment (message or file) default from the $$EERP$$ definition. Eerp acknowledgment (message) Eerp acknowledgment (file)	RFIC/$$EERP$$

Forwarding a Request

This request refers to the reception initial request. Only parameter #1 is provided. /DSN, /P99, /LAB are invalid as these information are retreived in the RENC information for the initial request.

Argument	Field	Description	Required or default
#1	FUN	Function - F=Forward	Required
	REQ	Request number, 8 alphanumeric characters. Example: /REQ=09800005	Required
	SPN	Remote partner name (adjacent)	Required
	SID	Local name (alias)	RPAR/Sysin
	PSW	Local password (alias)	RPAR/Sysin
	NTF	Notification option	RFIC
	PRT	Priority	RFIC
	LNK	Link type	RPAR
	DAT	Scheduling date	Immediat
	FAG	File agent option	N

Forwarding a Transfer Definition

This request provides the initial request information. No /REQ= parameter is provided.

Argument	Field	Description	Required or default
#1			
	FUN	Function - F=Forward	Required
	SPN	Remote partner name (adjacent)	Required
	SID	Local name (alias)	RPAR/Sysin
	PSW	Local password (alias)	RPAR/Sysin
	NTF	Notification option	RFIC
	PRT	Priority	RFIC
	LNK	Link type	RPAR
	DAT	Scheduling date	Immediat
	FAG	File agent option	N

Argument	Field	Description	Required or default
#2		Transfer definition	Required
	ORG	Origine of transfer. 1 to 8 alphanumeric characters. (pi3)Example: /ORG=Orgtrf01	Required
	DST	Destination of transfer. 1 to 8 alphanumeric characters. (pi4)Example: /DST=DSTtrf01	Required
	P11	File type. 4 hexadecimal characters. (Pi11) Example: 01FA	Required
	P12	File name. 1 to 8 alphanumeric characters. (pi12) – RFIC definition. Example: /P12=Ftest01	Required
	P13	Transfer identification. 1 to 8 numeric characters. (pi13) Example /P13=18	Required
	P51	File creation date: 12 numeric characters. Example: /P51=040110092503	Required
	P61	Transfer sender: 0 to 24 characters. (pi61) Example: /P61=Client name	Required
	P62	Transfer receiver: 0 to 24 characters. (pi62) Example: /P62=Service name	Required
#3, #4, #5			
DSN		Physical file name	RFIC
P99		User data	RFIC
LAB		File label	

Chapter 3

Exits and Procedures

This chapter provides an overview of exits and user procedures for start and end of transfer, and describes how to implement them.

About User Exits

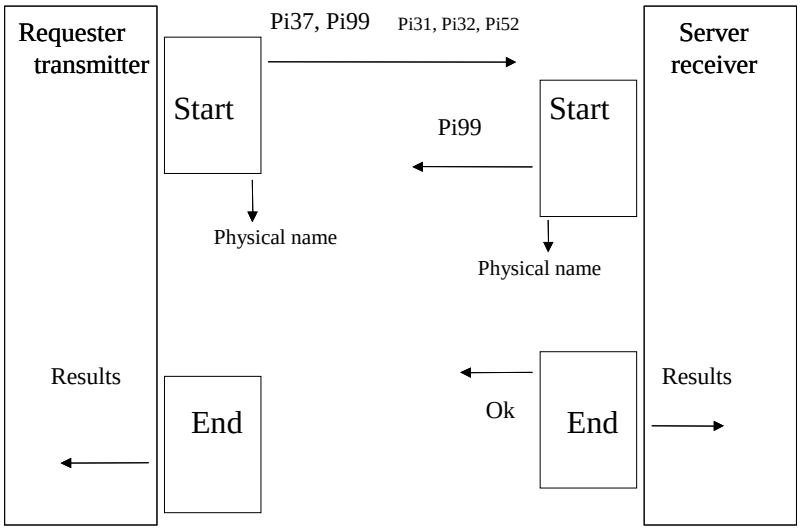
An exit is an executable program in C language that is synchronous, that is, it runs synchronously with the transfer. This distinguishes an exit from an integrated user procedure which runs asynchronously, or separate from the transfer. The following table describes the functions available using the exit interface.

Function	Description
Managing physical file names	At the start of transfer, the exit can be used to determine the physical name of the file. This name is normally determined by Connect:Express based on the symbolic file definition, but it can be necessary to determine the name at the time of transfer. The exit receives the name determined by Connect:Express and Connect:Express retrieves this modified field from the exit.
Processing the PeSIT fields	The communication mechanism between the exit and Connect:Express enables you to process the Pi99 and Pi37 fields at the start of transfer or to process them at the end of transfer. You can also use this exit to modify other PeSIT parameters such as, Pi3 bis, Pi4 bis, Pi31, Pi32, Pi52 and the physical file name.
Monitoring transfers	The end of transfer exit enables you to control transfer operations using the return codes and transfer statistics, such as number of bytes, number of records, and duration.

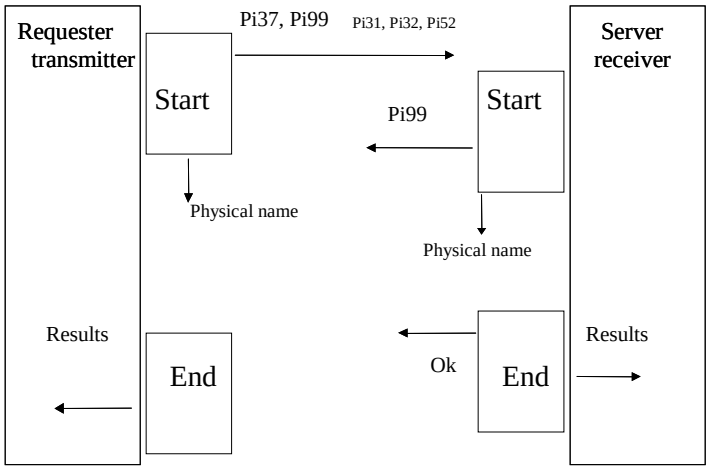
The exit receives a structure called d1b8ruex.h that provides information about the transfer at the start of transfer, and is completed at the end of transfer with the transfer results and the return codes.

The following diagrams show the stages of a PeSIT transfer for a transmission request and a reception request.

Transmission Request



Reception Request



With a PeSIT transfer, the start of transfer exit enables you to process the Pi37 on the transmitter's side, as well as the Pi31, 32, and 52 that describe the transmitted file. This exit also enables you to process the Pi99, which is a free field when the partner is type Other.

The start of transfer exit enables you to determine the local physical file name to transfer for any protocol, and the end of transfer exit enables you to validate the transfer on the receiver side for any protocol. This exit receives the results for both successful transfers and failed transfers.

Implementing User Exits

User exits must be saved in the exit directory. Connect:Express can activate user exits at the start or end of a transfer. You use the symbolic file definition to specify which exit to release at the start or end of transfer, in transmission or reception. The following screen shows the second screen of the symbolic file definition.

```
C:E / UNIX 145  -------------------- FILES DIRECTORY ------------------------ TOM1
OPTION ===>
SYMBOLIC NAME        : FICHIER DEFINITION : D DIRECTION : R
TRANSMISSION :
START EXIT ............ : EXTDEBT.....
START COMMAND ......... : ............
END EXIT .............. : EXTENDT.....
END COMMAND ........... : ............
RECEPTION :
START EXIT ............ : EXTDEBR.....
START COMMAND ......... : ............
END EXIT .............. : EXTENDT.....
END COMMAND ........... : ............
DO YOU WANT TO GO ON ? UPD : 20030722112010 C:E 142
-ENTER- NEXT FIELD -F3- CANCEL -F8- COMPLETION
```

The following table lists the components provided in the exit directory of Connect:Express.

Module	Description
d1b8uex.h	Communication structure between a user exit and Connect:Express.
chpi37.c	Source file: Example of loading the Pi37 field
user.c	Source file: displays transfer parameters

Connect:Express launches the exit with a parameter structure called d1b8ruex.h, written in a temporary file. The name of the temporary file is passed in a parameter to the exit. The structure d1b8ruex.h is located in the /exit directory. Refer to Appendix A for an example of this file.

If the STRACE parameter is set to 1 in the Connect:Express SYSIN file, a trace file is created in the /exit directory. The file name has the format Ex_QQQnnnnn. The x variable has the value I when the exit is called at the start of transfer, E when the exit is called at the end of a successful transfer, and F when the exit is called at the end of an interrupted transfer.

When the user exit is called at the start of the transfer, it can modify the following values: the physical file name to transmit, and the PeSIT fields Pi99, Pi37, Pi52, Pi31, Pi32, Pi32, Pi61, Pi62, Pi3, and Pi4 bis.

Connect:Express waits for the results of the exit before continuing to process the transfer. At the start of transfer, it recovers the new values (Pi31, Pi32, Pi37, Pi52, Pi99, Pi61, Pi62, Pi3 bis, and Pi4 bis) before using them and sending them to the remote partner. At the end of transfer, the exit can cause the transfer to be rejected.

Processing with a User Exit (Example)

The following screen shows an example of a simple exit. This exit reads a parameters file, modifies it, and then rewrites the parameters file.

```
/* ************************************
   * Example of an exit for loading the PI37 *
   ************************************ */
#include <stdio.h>
#include <errno.h>
#include "d1b8ruex.h"

#define SIZE_D1B8 sizeof(struct d1b8ruex)

struct d1b8ruex d1b8;

FILE *param;/* File Pointer to parameters file */
int bytes;/* To check File size              */
int i;

main(int argc, char *argv[])
{
param = fopen(argv[1], "r+");
if (param == NULL) {
  perror("");
  printf("Error Opening %s File.\n", argv[1]);
  exit(2);
}

bytes = fread((char *)&d1b8,1,SIZE_D1B8,param);
if (bytes!=SIZE_D1B8) {
  perror("");
  printf("Error Reading %s File.\n", argv[1]);
  exit(3);
}
/* ************************************************
   * Loading a complete file name in the PI37 *
   ************************************************ */
 strncpy((char *)d1b8.label,"C:\\CXV301\\FICHIER\\NOMFICHIER",80);

/* Rewriting the temporary file TOM */
fseek(param,0,0);
bytes = fwrite((char *)&d1b8,1,SIZE_D1B8,param);
if (bytes!=SIZE_D1B8) {
  perror("");
  printf("Error Writing %s File. %d bytes written instead of %d\n",
    argv[1],bytes,SIZE_D1B8);
  exit(3);
}

fclose(param);
exit(0);
}
```

About User Procedures

In contrast with exits, an integrated user procedure is launched by the monitor and executes outside of transfer operations, or asynchronously. It is a shell procedure that enables you to run a process during transfer operations.

The symbolic file definition enables you to integrate user procedures when a transfer executes. You can activate a user procedure at the start or end of reception or at the start or end of transmission. An end of transfer procedure is only activated when the transfer is successful.

The general procedure UEXERR provided with Connect:Express UNIX enables you to manage incidents or errors independently of the symbolic file definition. It is activated when a transfer fails during the connection phase or when the transfer is interrupted. You must modify the contents of the procedure, but the procedure name is fixed.

```
General Procedure: UEXERR
```

The following table describes the available functions using integrated user procedures.

Function	Description
Processing associated with the transfer	You can specify a user procedures in the symbolic file definition to activate it at the beginning or end of transfer, for example.
Managing errors and incidents	You can manage network and transfer incidents globally using the UEXERR procedure. This general procedure is activated for each incident during the connection phase. You associate an end of transfer command to this procedure. The UEXERR procedure delivered with Connect:Express must be customized.

Implementing User Procedures

You must save user procedures in the /exit directory.

Connect:Express can run user procedures at the start or end of transfer. The symbolic file definition enables you to specify which command you want to run at the start or end of transfer, in transmission or reception. The following screen shows the second screen of the symbolic file definition.

```
C:E / UNIX 145  -------------------- FILES DIRECTORY ------------------------ TOM1
OPTION ===>
SYMBOLIC NAME        : FICHIER DEFINITION : D DIRECTION : R
TRANSMISSION :
START EXIT ............ : ............
START COMMAND ......... : CMDDEBT.....
END EXIT ............... : ............
END COMMAND ........... : CMDENDT.....
RECEPTION :
START EXIT ............ : ............
START COMMAND ......... : CMDDEBR.....
END EXIT ............... : ............
END COMMAND ........... : CMDENDR.....
DO YOU WANT TO GO ON ? UPD : 20030722112010 C:E 142
-ENTER- NEXT FIELD -F3- CANCEL -F8- COMPLETION
```

The following table lists the components provided in the /exit directory.

Module	Description
UEXEERP	Procedure for end to end acknowledgment.
UEXERR	General procedure for error management.

Module	Description
UEXFWRD	Procedure for end to end forward process.
TRFOK	Example: Displays parameters
ROUTAGE	Example: Routing comand shell procedure
UEXROUT	General procedure for routing comand shell procedure
ROUTPI62	Example: Routing comand shell procedure

Parameters

Connect:Express launches the user procedure with a group of parameters numbered 1 to 22, and then continues the processing without waiting for the results. The following parameters are passed to the user procedure.

Parameter	Contents
$1	Request number
$2	Symbolic file name
$3	Symbolic partner name
$4	Physical file name
$5	Transfer direction
$6	System return code (SRC)
$7	Connect:Express return code (TRC)
$8	Protocol return code (PRC)
$9	Pi99 received
$10	Pi99 sent
$11	Origin of transfer
$12	Transfer destination
$13	Local name
$14	Label
$15	Sender of the file
$16	Receiver of the file
$17	Request start date
$18	Request start time
$19	Transfer date
$20	Julian date
$21	Number of records
$22	KBytes
$23	Request end date

Parameter	Contents
$24	Request end time
$25	Type of request (N/I/H/M/E)
$26	File type (4 hexadecimal)
$27	Transfer identification (8 numeric)
$28	File date-time (12 numeric)

Processing with an Integrated User Procedure (Example)

The following screen displays an example of a user procedure with parameters received as input:

```
#
# SHELL Command TRFOK
#
REQ="$1"
FIC="$2"
PART="$3"
NOM_PHY="$4"
SENS="$5"
SRC="$6"
TRC="$7"
PRC=="$8"
PI99R="$9"
shift
PI99S="$9"
shift
ORG="$9"
shift
DST="$9"
shift
LOC="$9"
shift
LAB="$9"
shift
PI61="$9"
shift
PI62="$9"
shift
RSD="$9"
shift
RST="$9"
shift
ETA="$9"
shift
QQQ="$9"
shift
NREC="$9"
shift
KBYT="$9"
shift
RED="$9"
shift
RET="$9"
echo "TRANFER TERMINATED WITHOUT PROBLEM"
echo "REQUEST              $REQ"
echo "FILE NAME            $FIC"
echo "PARTNER NAME         $PART"
echo "PHYSICAL NAME        $NOM_PHY"
echo "TRANSFER DIRECTION   $SENS"
echo "SYSTEM RETURN CODE   $SRC"
echo "TOM RETURN CODE      $TRC"
echo "PROTOCOL RETURN CODE $PRC"
echo "PI99 RECEIVED        $PI99R"
echo "PI99 SENT            $PI99S"
echo "TRANSFER ORIGIN      $ORG"
echo "TRANSFER DESTINATION $DST"
echo "LOCAL NAME           $LOC"
echo "LABEL                $LAB"
```

Continued

```
echo "USER SENDER (PI61) $PI61"
echo "USER RECEIVER (PI62) $PI62"
echo "REQUEST START DATE $RSD"
echo "REQUEST START TIME $RST"
echo "REQUEST TRANSFER STATE $ETA"
echo "JULIAN DATE $QQQ"
echo "NUMBER OF RECORDS $NREC"
echo "K. BYTES $KBYT"
echo "REQUEST END DATE $RED"
echo "REQUEST END TIME $RET"
```

Chapter 4

The TOM_PRM Command

This chapter describes the TOM_PRM command.

About the TOM_PRM Command

The tom_prm command enables you to manipulate parameters (partners, symbolic files and tables) in the Connect:Express database using text or binary files. The Connect:Express database is made up of the following files:

Database	File Name
Partners	RPAR.idx, RPAR.dat (Sequential indexed file)
Symbolic files	RFIC.idx, RFIC.dat (Sequential indexed file)
In progress	RENC.idx, RENC.dat (Sequential indexed file)
Session tables	RTAB (Binary file)
Presentation tables	RPRE (Binary file)

These files are located in the $TOM_DIR/config directory of the monitor.

WARNING: In the current version, you must stop the monitor before using the tom_prm command.

tom_prm gives you the option of extracting data from the Connect:Express database using a text or binary file, and then uploading data back to the database using these extraction files. In addition, tom_prm enables you to delete an entire database and recreate an empty database.

The tom_prm command can take as input:

- the name of a text file with instructions for extracting or uploading data.
- the name of a binary file with data to upload to the database.
- a directive followed by parameters.

Syntax of tom_prm

The executable tom_prm takes the following parameters:

```
$ tom_prm ?
$ tom_prm input=file name,[format=(TEXT|BINARY)]
$ tom_prm directive parameters-of-the-directive
```

Parameter	Description
Tom_prm ?	Displays help about the user output.
file name	Refers to the name of the file used as input for tom_prm. This file can be: - A text file containing a group of directives to apply. See Format of a Directive Text File. - A binary file containing data to update in the database. See Format of a Binary File. In this case, the parameter format=BINARY must be present in the command.
format	describes the format of an input file. format=TEXT is the default value.
directive	Is a simple directive name referenced in *Text Files Containing Directives* on page 4-4. The syntax enables you to specify a directive followed by parameters in the command line of the shell. In this case the parameters of the directive must be entered consecutively, separated by a comma and without spaces between each.

Using the TOM_PRM Command

You enter the TOM_PRM command at the UNIX command line. This section shows examples of how you can use the TOM_PRM command.

Extracting to a Text File

The following screen shows how you can use the tom_prm command to extract data to a text file.

```
$ tom_prm extract type=all,output=extract.txt
$ tom_prm extract type=partner,output=extract.txt,format=text
$ tom_prm extract type=all >extract.txt
```

In this example, the contents of the database is extracted to a text file containing a directive. These directives can be reused later by the tom_prm command to upload the corresponding elements back to the database. This file is readable and can be modified using any text editor.

You can also extract all parameters in the database to a text file. See *Extracting all Parameters from the Database* on page 4-11 for an example.

Extracting to a Binary File

The following screen shows how you can use the tom_prm command to extract to a binary file.

```
$ tom_prm extract type=all,output=extract.bin,format=binary
$ tom_prm extract type=partner,output=extract.bin,format=binary
$ tom_prm extract type=all,format=binary >extract.bin
```

In this example, the selected elements from the database are extracted to a binary file. This file can then be used by another program or used to upload data back to the database.

Reinitializing the Database

The following screen shows how you can use the tom_prm command to reinitialize the database.

```
$ tom_prm database mode=delete
$ tom_prm database mode=create

or

$ tom_prm input=init.cmd

The file init.cmd is a text file containing the following 2 directives:

database mode=delete
database mode=create
```

In this example, the database files are deleted and then recreated with no data.

Uploading Data to the Database from a Text File

The following screen shows how you can upload data to the database from a text file

```
$ tom_prm input=extract.txt
```

In this example, the file called extract.txt contains a directive to upload data. This file extract.txt could have been created directly with a text editor or generated by the extract command.

Uploading Data to the Database from a Binary File

The following screen shows how you can upload data to the database from a binary file

```
$ tom_prm input=extract.bin,format=binary
```

In this example, the contents of the binary file extract.bin is uploaded to the database. This file, extract.bin, can be created by a program or generated by an extract command performed earlier.

Modifying an Element in the Database

The following screen shows the command to modify an element in the Connect:Express database.

```
$ tom_prm partner name=PART01,mode=delete
$ tom_prm file name=FILE01,lrec=1024,mode=replace
```

In this example, the partner PART01 is deleted from the database, the symbolic file FILE01is modified, and the Record Length field takes on the new value of 1024.

User out and Error out

The tom_prm command returns 0 when no error is encountered, and a 1 if there was an error. Information about the execution of the command is routed to the Error Out of the shell.

The output in text format of an extraction command is directed by default to the User Output of the shell. The output can be directed to a file either by using the parameter output=file name or by redirecting the user exit to the shell.

The output in binary format of an extraction command is always directed to a file. (The parameter output=file name is required in this case.)

Text Files Containing Directives

Each directive has the format:

```
DIRECTIVE param1=value1,… , paramN=valueN
```

You can insert comments in the text. A comment begins with /* and ends with */. The comments and the directives can carry over more than one line, but param=value cannot be placed on two different lines. Use a comma as a separator between each parameter in a directive.

The following table lists the type of directive files that you can create.

Directive	Description
DATABASE	Enables you to delete an entire database or create a new, empty one.
PARTNER	Enables you to create, replace or delete a partner.
FILE	Enables you to create, replace or delete a symbolic file.
SESSION	Enables you to replace a session table.
PRESENTATION	Enables you to replace a presentation table.
EXTRACT	Enables you to extract all or part of a database.

This section describes the detailed syntax for each directive type. The syntax of directive names and parameter names is not case sensitive.

DATABASE Directive

The Database directive enables you to reinitialize the database CTREE.

Directive	Parameters	Description
DATABASE	MODE=CREATE	Creates empty files RPAR, RFIC and RENC according to the contents of the file database.p
	MODE=DELETE	Deletes the files RPAR, RFIC, RENC, RTAB and RPRE in $TOM_DIR/config

WARNING: If you use this directive in Delete mode, it deletes the RENC file.

You can recreate a database and retain a RENC file by using the following procedure:

1. Stop the monitor.
2. Save the two files RENC.dat and RENC.idx that are located in $TOM_DIR/config.
3. Apply the directives DATABASE DELETE then CREATE.
4. Restore the saved RENC files.
5. Restart the monitor.

PARTNER Directive

This directive enables you to create, delete or update partner information.

Directive	Parameters	Description
PARTNER	NAME=name	Up to 8 characters
	MODE=mode	CREATE, REPLACE or DELETE
	PASSWD=password	Up to 8 characters
	STATE=state	E: Enabled, H: Held
	TYPE=type-of-partner	T: TOM O: Other
	PROT=protocol-number	1: Etebac3, 2: FTP, 3: PeSIT
	SESSION=session-table-number	1 -> 9
	MAXSES=maximum-number-of-sessions	2 numeric characters
	MAXSESIN=maximum-number-of-incoming-sessions	2 numeric characters
	MAXSESOUT=maximum-number-of-outgoing-sessions	2 numeric characters
	LINK=link-type	X: X25, P: PAD, T:TCPIP, M: X25 and TCPIP
	X25PORT=x25-device-number	0 -> 3
	RDIALNO=x25-called-address	Up to 15 numeric characters
	LDIALNO=x25-calling-address	Up to 15 numeric characters
	DATA=x25-user-data	Up to 8 hexadecimal characters
	LOCFAC=x25-facilities	Up to 16 hexadecimal characters
	HOST=TCP-host-name	Up to 32 characters
	TCPPORT=TCP-port-number	Up to 5 numeric characters
	TCPADDR=IP-Address	Ip address (dotted format)
	DPCSID=dpcsid	Up to 8 characters
	DPCPSW=dpcpsw	Up to 8 characters

Directive	Parameters	Description
	FTPFILE=default-ftp-file-name	Up to 8 characters
	NRETRY=number of retries	Up to 2 numeric characters
	INTSESST=interval-session-timer	Up to 2 numeric characters
	INTTRANST=interval-transfer-timer	Up to 2 numeric characters
	SSLPARM=sslparm ID	Up to 8 characters
	CTRLDN=ctrldn ID	Up to 8 characters

FILE Directive

This directive enables you to create, delete, or update symbolic file information.

Directive	Parameters	Description
FILE	NAME=name	Up to 8 characters
	MODE=mode	CREATE, REPLACE or DELETE
	STATE=state	E: Enabled, H: Held
	DIRECTION=transfer-direction	T:Transmit, R:Receive, *:Both
	RPART=receiving-partner	Up to 8 characters
	TPART=transmitting-partner	Up to 8 characters
	PRIORITY=transfer-priority	0:Urgent,1:Fast,2:Normal
	DEFTYPE=definition-type	D:Dynamic, F:Fixed
	PRESENTATION=presentation-table	1 -> 9
	FICPARAMS=use-of-ficparms-file	Y,N
	SPACE=space-to-reserve-flag	Y,N
	ALLOCATION=allocation-rule	0:Indifferent,1:Preallocated,2:To create
	DSN=physical-name	Up to 44 characters
	FORMAT=record-format	TF,TV,BF,BV,T*,B*
	LREC=record-length	Up to 5 numeric characters
	REMDSN=remote-dsn	Up to 44 characters
	FTPTYPE=ftp-type	A:Ascii,E:Ebcdic,B:Binary,*:Unchanged
	FTPSTRUCT=ftp-structure	F:File,R:Record,*:Unchanged
	FTPMODE=ftp-mode	B:Block,S:Stream,*:Unchanged
	FTPSTOU=ftp-store-unique-flag	Y,N
	SSEXIT=send-start-exit	Up to 12 characters
	SEEXIT=send-end-exit	Up to 12 characters

Directive	Parameters	Description
	SSCMD=send-start-cmd	Up to 12 characters
	SEECMD=send-start-cmd	Up to 12 characters
	RSEXIT=receive-start-exit	Up to 12 characters
	REEXIT=receive-end-exit	Up to 12 characters
	RSCMD=receive-start-cmd	Up to 12 characters
	REECMD=receive-start-cmd	Up to 12 characters
	FA=file-agent-flag	Y,N
	NOT=notification	Space/0/1/2/3/4/5/6/7

SESSION Directive

This directive enables you to create, delete, or update session table information.

Directive	Parameters	Description
SESSION	NAME=table-number	1 -> 9
	MODE=mode	REPLACE
	MSGSIZE=line-message-size	Up to 5 numeric characters
	SYNC=synchronization-in-Kbytes	Up to 5 numeric characters <65
	WINDOW=synchronization-window	Up to 2 numeric characters
	LEVEL=protocol-version-level	1:PeSIT D,2:PeSIT E
	RETRY=max-number-of-retries	Up to 2 numeric characters
	CRC=use-CRC-flag	Y,N

PRESENTATION Directive

This directive enables you to create, delete or update presentation table information.

Directive	Parameters	Description
PRESENTATION	NAME=table-number	1 -> 9
	MODE=mode	REPLACE
	COMP=type-of-compression	0:No,1:Horizontal,2:Vertical,3:Mixed
	MULTART=multi-article-flag	Y,N
	TRANSLATION=translation-table-number	0 -> 9

EXTRACT Directive

This directive enables you to extract specific data from the Connect:Express database. The output can be redirected to a text file using the parameter OUTPUT=file name, or by using shell redirection. Otherwise, the output is directed to a user exit.

The value of the parameter NAME can contain wildcard characters like * and ?. TYPE=ALL indicates all types of objects (partner, file, session, presentation) for the value of NAME.

Directive	Parameters	Description
EXTRACT	TYPE=type-of-data	ALL,PARTNER,FILE,SESSION,PRESENTATION
	NAME=name-of-object-to-extract	Up to 8 characters
	OUTPUT=output-filename	Up to 255 characters
	FORMAT=output-format	TEXT or BINARY

The following table shows some examples:

Example	Result
EXTRACT TYPE=ALL	Extract all the parameters. The result is displayed as standard user output.
EXTRACT TYPE=PARTNER,NAME=PART*	Extracts the definition of all partners whose name begins with PART*. The result is displayed as standard user output.
EXTRACT TYPE=ALL,NAME=*01,OUTPUT=EXT.PARM	Extracts the definition of all objects whose name ends in “01 “. The result is sent to the file EXT.PARM in a text format.
EXTRACT TYPE=ALL,OUTPUT=extract.bin,FORMAT=BINARY	Extracts all the parameters. The result is saved in binary format in the file extract.bin.

Binary Extract Files

This section describes the format of binary files generated by extraction commands. Each record is separated by the LineFeed character. (0x0A).

The first character of each record is a letter that describes the type of data contained in the record.

P: Partner

F: Symbolic file

S: Session table

R: Presentation table

The record type character is followed by the corresponding binary structure, then the separator LF. The binary structures for the partners and symbolic files are defined in the file d0b8z20.h, located in the $TOM_DIR/itom directory.

The structures contain only alphanumeric characters that can be displayed by the STERM utility. The replacement character for each field is a space (SP).

Note: tom_prm does not verify the syntax when it uploads data to the database from a binary file. Therefore, if a binary extract file is modified by the program, the program must do its own syntax verification based on the syntax rules used by the STERM utility.

The following screens show the binary structures for a partner, symbolic file, session table, and presentation table:

Partner Structure

```
struct partenaire {
  char nom_sym[8];   /* Partner Symbolic Name              */
  char passwd[8];    /* Password                           */
  char etat_init[1]; /* Initialization State               */
  char nature[1];    /* Partner Type (T:CX or O:Other)     */
  char num_prot[1];  /* Protocol (1:ETB3,2:FTP,3:PeSIT)    */
  char tab_sess[1];  /* Session Table (1 -> 9)             */
  char port[1];      /* X25 Device                         */
  char nb_liai[2];   /* Number of Sessions                 */
  char typ_lia[1];   /* Type of Link (L, X, M)             */
  char num_rem[15];  /* X25 Remote Address                 */
  char num_loc[15];  /* X25 Local (Sub)Address             */
  char loc_fac[16];  /* Facilities                         */
  char udf[8];       /* User Data Filed                    */
  char upd_date[14]; /* Date of Last Update YY/MM/DD HH:SS  */
  char userid[8];    /* User who updates                   */
  char dpcsid[8];    /* DPCSID alias                       */
  char dpcpsw[8];    /* DPCPSW alias                       */
  char tcp_host[32]; /* Host Name                          */
  char tcp_addr[15]; /* Host Address                       */
  char tcp_port[5];  /* Host Port                          */
  char ftpfile[8];   /* FTP Default File                   */
  char nb_liai_in[2];/* Number of Sessions IN              */
  char nb_liai_out[2]; /* Number of Sessions OUT             */
  char nb_repr[2];     /* Number of retries                  */
  char int_sess[2];    /* Interval session time              */
  char int_trans[2];   /* Interval tranfer time              */
  char sslparmid[8]; /* Ssl transfer profile               */
  char ctrldn[8];    /* Ctrldn id                          */
  char filler[54];   /* For Future Use                     */
};
```

Symbolic File Structure

```
struct fichier {
  char nom_sym[8];/* Symbolic File Name                */
  char etat_init[1];  /* Initialization State              */
  char direction[1];  /* Direction                         */
  char recepteur[8];  /* Receiver Partner                  */
  char emetteur[8];   /* Sender Partner                    */
  char priorite[1];   /* Priority                          */
  char typ_def[1];    /* Definition D:Dynamic,F:Fixed      */
  char present[1];    /* Presentation Table 1->9           */
  char nom_phy[44];   /* Dsname                            */
  char format[2];     /* TV, TF, BF, BU                    */
  char record[5];     /* Record Length                     */
  char exit_de[12];   /* Exit/Start of transfer/Sender     */
  char comm_de[12];   /* Command/Start of transfer/Sender  */
  char exit_fe[12];   /* Exit/End of transfer/Sender       */
  char comm_fe[12];   /* Command/End of transfer/Sender    */
  char exit_dr[12];   /* Exit/Start of transfer/Receiver   */
  char comm_dr[12];   /* Command/Start of transfer/Receiver */
  char exit_fr[12];   /* Exit/End of transfer/Receiver     */
  char comm_fr[12];   /* Command/End of transfer/Receiver  */
  char upd_date[14];  /* Date of Last Update YY/MM/DD HH:SS */
  char userid[8];     /* User who updates                  */
  char remotedsn[44]; /* Remote DSN                        */
  char tsm[3];        /* Type/Structure/Mode               */
  char tab[1];        /* Optional Table                    */
  char rule[1];       /* Allocation Rule 0/1/2             */
  char alloc[1];      /* Allocation Flag Y/N               */
  char stou[1];       /* Store Unique Flag Y/N FTP         */
  char fa[1];         /* Flag File agent Y/N               */
  char notif[1];       /* Notification: space/1/2/3         */
  char filler[5];     /* For Future Use                    */
};
```

Session Table Structure

The session tables are named 1 through 9.

```
struct session {
  char table[1];   /* Table name ('1', ... ,'9')           */
  char msgsize[5]; /* Line message size                    */
  char synchro[2]; /* Synchronization in KBytes            */
  char window[2];  /* Synchronization window               */
  char level[1];   /* Protocol version level               */
  char retry_nb[2];/* Maximum number of retries            */
  char upd_date[14]; /* Date of last update YY/MM/DD HH:SS    */
  char userid[8];  /* Name of the user updating the record  */
  char crc[1];     /* CRC Y/N                              */
  char filler[4];
};
```

Presentation Table Structure

The presentation tables are named 1 through 9.

```
struct presentation {
  char table[1];/* Table name ('1', ... ,'9')           */
  char compression[1];/* Compression '0','1','2' or '3'       */
  char multiart[1];/* Multi-article flag                    */
  char translat[1];/* Translation table number              */
  char filler[10];
  char upd_date[14];/* Date of last update YY/MM/DD HH:SS   */
  char userid[8];/* Name of the user updating the record */
  char filler[6];
};
```

Extracting all Parameters from the Database

The following command enables you to extract all of the parameters from the database.

```
$ tom_prm extract type=all,output=db.parm
```

The results are extracted to the file db.parm, as shown in the following example:

```
FILE
                  NAME         = BOUCLE,
                  STATE        = E,
                  DIRECTION    = *,
                  RPART        = $$ALL$$,
                  TPART        = $$ALL$$,
                  PRIORITY     = 0,
                  DEFTYPE      = D,
                  PRESENTATION = 1,
                  FICPARAMS    = N,
                  SPACE        = N,
                  ALLOCATION   = 0,
                  DSN          = $TOM_DIR/in/&REQNUMB.tmp,
                  FORMAT       = TV,
                  LREC         = 08192,
                  REMDSN       = ,
                  FTPTYPE      = *,
                  FTPSTRUCT    = *,
                  FTPMODE      = *,
                  FTPSTOU      = N,
                  SSEXIT       = ,
                  SEEXIT       = ,
                  SSCMD        = ,
                  SECMD        = ,
                  RSEXIT       = ,
                  REEXIT       = ,
                  RSCMD        = ,
                  RECMD        = ,
                  FA           = N,
                  NOT          = ,
                  MODE         = REPLACE
```

```
FILE
                NAME         = ...,
                ......etc...........
PARTNER
                NAME         = BOUCLE,
                PASSWD       = PSW,
                STATE        = E,
                TYPE         = O,
                PROT         = 3,
                SESSION      = 1,
                X25PORT      = ,
                MAXSES       = 20,
                MAXSESIN     = 12,
                MAXSESOUT    = 8,
                LINK         = T,
                RDIALNO      = ,
                LDIALNO      = ,
                DATA         = ,
                LOCFAC       = ,
                HOST         = ,
                TCPADDR      = 192.168.0.33,
                TCPPORT      = 06677,
                DPCSID       = BOUCLE,
                DPCPSW       = PSW,
                FTPFILE      = ,
                NRETRY       = ,
                INTSESST     = ,
                INTTRANST    = ,
                SSLPARM      = ,
                CTRLDN       = ,
                MODE         = REPLACE
PARTNER
                NAME         = ...,
                ......etc...........


SESSION
                NAME         = 1,
                MSGSIZE      = 08192,
                SYNC         = 32021,
                WINDOW       = 02,
                LEVEL        = 1,
                RETRY        = 09,
                CRC          = N,
                MODE         = REPLACE
SESSION
                NAME         = 2,
                ......etc...........
SESSION
                NAME         = 9,
                MSGSIZE      = 02048,
                SYNC         = 16011,
                WINDOW       = 01,
                LEVEL        = 1,
                RETRY        = 05,
                CRC          = ,
                MODE         = REPLACE
```

```
PRESENTATION
                NAME         = 1,
                COMP         = 0,
                MULTART      = Y,
                TRANSLATION  = 0,
                MODE         = REPLACE
PRESENTATION
                NAME         = 2,
                ......etc...........
PRESENTATION
                NAME         = 9,
                COMP         = 3,
                MULTART      = Y,
                TRANSLATION  = 0,
                MODE         = REPLACE
```

Appendix A

Structure Files

This appendix provides examples of the structure files, d0b8z20.h and d1b8ruex.h.

d0b8z20.h File

The compilation of the user program must include the option -L $TOM_DIR/itom. The environment variable corresponds to the root directory of the monitor, for example /home/tom1).

The d0b8z20.h structure is comprised of a header with substructures, as shown in the following diagram. The d0b8z20.h structure file begins with the PeSIT and network transfer parameters (st_trf), the innermost substructure. This is the reverse of the structure represented in the following diagram.

- *ZREQ_tom* Request parameters = Header **+ Request**
  - *uni_sci* Request = **File // Partner // Transfer request // Saving the RENC file**
    - *File :* Description of a File
    - *Partner* : Description of a Partner
    - *st_sci* : Parameters of a Transfer Request
    - *s_renc* : Saving the RENC file = Process status **+ Transfer parameters**
      - *st_trf* : Transfer parameters = **PeSIT parameters + Network parameters**
        - *s_pi* : PeSIT parameters
        - Network parameters : *s_x25* : X25 parameters **//** *s_tcp* : TCP/IP parameters

The following screen shows the d0b8z20.h file.

```
/* Network Structures */
struct s_x25_param {
  char appelant[15];      /* Calling DTE Number            */
  char appele[15];        /* Called DTE Number             */
  char port[1];           /* Device Name                   */
  char applid[8];         /* Routing List(IBM)/LAID (NCR)  */
  char command[1];      /* Error Command           */
  unsigned char loc_fac[8];  /* Facilities Field              */
  unsigned char udf[4];      /* User Data Field               */
  int nrc;              /* Network Return Code           */
  short lg_appelant;         /* Calling DTE Number Length     */
  short lg_appele;           /* Called DTE Number Length      */
  short lg_applid;           /* Routing List /LAID Length     */
  short loc_fac_lg;          /* Facility Field Length         */
  short udf_lg;              /* User Data Field Length        */
  unsigned char cause[2];    /* TRANSPAC Cause                */
  unsigned char diagnostic[2]; /* TRANSPAC Diagnostic         */
};

#define S_X25 sizeof(struct s_x25_param)

struct s_tcp_param {
  char port[5];              /* Port Service                  */
  char adresse[15];          /* Partner Internet Addresse     */
  char host[32];             /* Host Name                     */
  char command[1];           /* Error Command            */
  int nrc;   /* Network Return Code           */
  char filler[S_X25 - 53 - sizeof(int)];
};
#define S_TCP sizeof(struct s_tcp_param)

struct s_pi {
  char diag[3];      /* Diagnosticpi2  */
  char ident[8];     /* Identity of requesterpi3  */
  char idser[8];     /* Identity of server pi4  */
  char cac[8];       /* Access control     pi5  */
  char ver[1];       /* Number of Protocol version pi6  */
  char opo[3];       /* Synchro optionpi7  */
  char tyf[2];       /* File Type    pi11 */
  char nof[8];       /* Symbolic filename pi12 */
  char idt[3];       /* Transfer identificator pi13 */
  char atd[1];       /* Attributs askedpi14 */
  char trr[1];       /* Restart flag pi15 */
  char cod[1];       /* Data codepi16 */
  char prt[1];       /* Priority of transfert pi17 */
  char por[3];       /* Point of restart pi18 */
  char cft[1];       /* End of transfer code pi19 */
  char nps[3];       /* Number of synchronization pointpi20 */
  char cpr[2];       /* Compressionpi21 */
  char tac[1];       /* Access type pi22 */
  char res[1];       /* Resynchronizationpi23 */
  char mlt[2];       /* Multi-fpdu lengthpi25 */
  char nb_oct[4];    /* Number of bytes pi27 */
  char nb_art[4];    /* Number of articlespi28 */
  char far[1];       /* Format of articlepi31 */
  char loa[2];       /* Length of articlepi32 */
  char orf[1];       /* File organizationpi33 */
  char label[80];    /* Label du Fichier pi37 */
  char vur[1];       /* Unit of space allocationpi41 */
  char vme[4];       /* Maximum space allocation pi42 */
```

Continued

```
    char dhc[12];         /* Unused pi51 */
    char dhd[12];         /* Unused pi52 */
    char nom_phy[44];     /* File physical name*/
    char oc5[1];          /* Identity of receiver */
    char oc6[1];          /* Identity of transmitter*/
    char s trace[1];     /* Y to request trace activation. N to request deactivation */
    char no_req[8];       /* Request number */
    char typ_req[1];      /* Request Type (Normal or Inquiry)    */
    char typ_part[1];     /* Partner type (T or O)*/
    char userid[8];       /* Requester Userid*/
    char format[2];       /* Record Format (Text, Binary, Fixed, */
                          /* Variable Undefined) TF TV BF BU*/
    char etat[1];         /* Transfer State                 */
                          /* 0 : Waiting ('A')*/
                          /* 1 : Connected       */
                          /* 2 : Selected       */
                          /* 3 : Opened         */
                          /* 4 : Suspended      */
                          /* 5 : Transfered      */
                          /* 6 : Deselected            */
    char s_dsname[44];    /* Sender Physical Name                */
    char r_dsname[44];    /* Receiver Physical Name              */
    char multiart[1];     /* Flag MultiArticle (Y/N)             */
    char crc[1];          /* Crc Flag (Y/N)                      */
    char s_no_req[4];     /* Sen. Req. Number (Bin)              */
    char s_req_dat[4];    /* Sen. Req. Date (Bin)                */
    char s_group[8];      /* Sender User Group                   */
    char s_userid[8];     /* Sender User Id                      */
    char s_old_psw[8];    /* Sender Old Password                 */
    char s_new_psw[8];    /* Sender New Password                 */
    char rem_trc[4];      /* Remote TRC                          */
    char translat[1];     /* Translation Table                   */
    short trc;            /* TOM Return Code           */
    short src;            /* System Return Code*/
    short lg_ident;       /* Length of field 'ident'*/
    short lg_idser;       /* Length of field 'idser'*/
    short lg_fic;         /* Length of field 'nof'*/
    short support;        /* Type of link*/
    int ack_pos;          /* Offset of last acquitted synchro point*/
    char num_prot[1];     /* Protocol Number (1:Etebac3, 3:PeSIT)*/
    char password[8];     /* Password of Partner                 */
    char api[88];         /* API Field                           */
    char tsm[3];          /* Type/Structure/Mode FTP             */
    char alloc[1];        /* Allocation Flag Y/N                 */
    char rule[1];         /* Allocation Rule 0,1,2               */
    char stou[1];         /* Sore Unique Flag Y/N FTP            */
    char fa[1];           /* flag File agent Y/N                 */
    char ack_pos_lfs[8];  /* 64 bits (Large file support)          */
    char nb_oct_lfs[8];   /* 64 bits (Large file support)          */
    char s_pi99_254[254]; /* Pi99 sent                            */
    char r_pi99_254[254]; /* Pi99 received                        */
    char user_org[8];     /* User Origin               pi3 bis    */
    char user_dst[8];     /* User Destination          pi4 bis    */
    short lg_user_org;    /* Length of field 'origin'             */
    short lg_user_dst;    /* Length of field 'destination'        */
    char user_snd[24];    /* User Sender pi61 */
    char user_rcv[24];    /* User receiver pi62 */
    short lg_user_snd;    /* Length of field 'sender' */
    short lg_user_rcv;    /* Length of field 'receiver' */
```

Continued

```
  char quant[3];                    /* Julian date */
  char n_rep[2];                    /* Number of retries allowed */
  char int_sess[2];                 /* Interval session timer */
  char int_trans[2];               /* Interval transfer time */
  char notif[1];                   /* Notification flag: space/0/1/2/3 */
  char sslparmid[8];               /* SSL transfer parameters */
  char sslrc[8];                   /* OpenSSL return code */
  char sslused[1];                 /* Ssl used flag */
  char flag_rout[1];                /* Flag rout */
  char filler[105];                /* For Future use */
  };
struct st_trf {
  struct s_pi pi;
  union {
    struct s_x25_param x25_param;
    struct s_tcp_param tcp_param;
  } u_netw_param;
};
/* Definitions facilitant l'acces aux zones X25 et TCPIP */
#define X25   u_netw_param.x25_param
#define TCPIP u_netw_param.tcp_param

typedef struct s_renc  {  /* Record of RENC file.                    */
  char typ_dem[1];      /* 'D' Requester  'S' Server            */
  char nom_fic[8];      /* Logical name of file.                */
  char dat_sou[8];      /* Date of soumission, YYYYMMDD.        */
  char heu_sou[6];      /* Time of soumission, HHMMSS.          */
  char sen_tra[1];      /* Direction of transfer (T-Transmit,   */
                        /* R-Receive)                           */
  char eta_tra[1];      /* State of transfer :                  */
                        /*     C - in progress                  */
                        /*     A - awaiting selection           */
                        /*     D - defered                      */
                        /*     E - ended                        */
                        /*     O - abnormally ended             */
                        /*     J - automatic restart            */
                        /*     K - awaiting restart             */
  char dat_deb[8];      /* Date of start of transfer (YYYYMMDD).*/
  char heu_deb[6];      /* Time of start of transfer (HHMMSS).  */
  char dat_fin[8];      /* Date of end of transfer (YYYYMMDD).  */
  char heu_fin[6];      /* Time of end of transfer (HHMMSS).    */
  int nb_repr;          /* Number of retries (initialised to 0).*/
  char ori_com[1];      /* Origin of command:                   */
                        /*   S - TOM menu.                      */
                        /*   I - Application interface          */
                        /*   T - Batch                          */
  short ses;            /* Session Number which made the transfer   */
  int pid;              /* Process ID of STRF which transfered. */
  char support_origin;  /* Origin support of request            */
  char filler[13];      /* For Future Use                       */
  struct st_trf trfpar; /* Structure whith transfer parameters. */
} s_renc;
#define SIZE_RENC sizeof(struct s_renc)
```

Continued

```
struct st_sci {
  char dire[1];                    /* Direction                  */
  char file[8];                    /* Symbolic file name         */
  char part[8];                    /* Symbolic partner name      */
  char dsnam[44];                  /* Dsname                     */
  char prty[1];                    /* Priority                   */
  char dat[8];                     /* Date                       */
  char hour[6];                    /* Hour                       */
  char lnk[1];                     /* Link type                  */
  char udf[44];                    /* User data file             */
  char typ[1];                     /* Request type               */
  char sta[1];                     /* State of Request           */
  char dpcsid[8];                  /* Dpcsid for Alias           */
  char dpcpsw[8];                  /* Dpcpsw for Alias           */
  char format[2];                  /* Record Format TF TV BF BU  */
  char lrecl[5];                   /* Record Length              */
  char api[88];                    /* Api Field                  */
  char tsm[3];                     /* Type/Structure/Mode FTP    */
  char stou[1];                    /* Store Unique FTP           */
  char fa[1];                      /* flag File agent Y/N        */
  char label[80];                  /* Label du Fichier           */
  char s_pi99_254[254];            /* pi99 254 characters           */
  char user_org[8];                /* User Origin            pi3 bis */
  char user_dst[8];                /* User Destination       pi4 bis */
  char user_snd[24];               /* User Sender pi61              */
 char user_rcv[24];               /* User receiver pi62            */
 char quant_aa[2]                 /* Year AA for Julian date       */
 char quant[3]                    /* Julian date                   */
 char notif[1]                    /* Notification flag: space/0/1/2/3/4/5/6/7 */
 char noreq[8];                   /* request numberp1b8pe2e      */
 char dhc[12];                    /* File date Pi51p1b8pe2e      */
 char idt[8];                     /* Pi13p1b8pe2e                 */
 char ftype[4];                   /* Pi11p1b8pe2e                 */
 char filler[SIZE_RENC - 675];
 };
/*************************************************************************/
/*               Structure for displaying partner                        */
/*************************************************************************/
struct partenaire {
  char nom_sym[8];     /* Partner Symbolic Name              */
  char passwd[8];      /* Password                           */
  char etat_init[1];   /* Initialization State               */
  char nature[1];      /* Partner Type (TOM or Compatible)   */
  char num_prot[1];    /* Protocol Number (1->ETB3, 3->PeSIT) */
  char tab_sess[1];    /* Session Table (1 -> 9)             */
  char port[1];        /* X25 Device                         */
  char nb_liai[2];     /* Number of Sessions                 */
  char typ_lia[1];     /* Type of Link (L, X, M)             */
  char num_rem[15];    /* X25 Remote Address                 */
  char num_loc[15];    /* X25 Local (Sub)Address             */
  char loc_fac[16];    /* Facilites                          */
  char udf[8];         /* User Data Filed                    */
  char upd_date[14];   /* Date of Last Updating              */
  char userid[8];      /* Userid who updates                 */
  char dpcsid[8];      /* DPCSID alias                       */
  char dpcpsw[8];      /* DPCPSW alias                       */
  char tcp_host[32];   /* Host Name                          */
  char tcp_addr[15];   /* Host Adresse                       */
  char tcp_port[5];    /* Host Port                          */
  char ftpfile[8];     /* FTP Default File                   */
  char nb_liai_in[2];  /* Number of Sessions IN    15/01/01   */
  char nb_liai_out[2]; /* Number of Sessions OUT   15/01/01   */
   char nb_repr[2];     /* retry number                       */
```

Continued

```
  char int_sess[2];    /* connection timer                    */
  char int_trans[2];   /* request timer                       */
  char sslparmid[8];   /* SSL transfer parameters             */
  char filler[62];     /* For Future Use                      */
};
/*********************************************************************/
/*               Structure for displaying file                       */
/*********************************************************************/
struct fichier {
  char nom_sym[8];    /* Symbolic File Name   */
  char etat_init[1];  /* Initialization State */
  char direction[1];  /* Direction            */
  char recepteur[8];  /* Receiver Partner     */
  char emetteur[8];   /* Sender Partner       */
  char priorite[1];   /* Priority             */
  char typ_def[1];    /* Definition Dyn/Fix   */
  char present[1];    /* Presentation Table 1->9  */
  char nom_phy[44];   /* Dsname               */
  char format[2];     /* TV, TF, BF, BU       */
  char record[5];     /* Record Length        */
  char exit_de[12];   /* Exit/Comm Transm.    */
  char comm_de[12];
  char exit_fe[12];
  char comm_fe[12];
  char exit_dr[12];   /* Exit/Comm Reception  */
  char comm_dr[12];
  char exit_fr[12];
  char comm_fr[12];
  char upd_date[14];  /* Date of Last Updating*/
  char userid[8];     /* Userid who updates   */
  char remotedsn[44]; /* Remote DSN           */
  char tsm[3];        /* Type/Structure/Mode  */
  char tab[1];        /* Optionnal Table      */
  char rule[1];       /* Allocation Rule 0/1/2 */
  char alloc[1];      /* Allocation Flag Y/N */
  char stou[1];       /* Store Unique Flag Y/N FTP */
  char fa[1];         /* flag File agent Y/N */
  char notif[1]       /* Notification flag: space/0/1/2/3/4/5/6/7 */;
  char filler[5];     /* For Future Use       */
};

/*********************************************************************/
/* Union                                                             */
/*********************************************************************/
union uni_sci {
  struct st_sci zreq_tom_sci;
  struct s_renc zreq_tom_renc;
  struct partenaire zreq_tom_part;
  struct fichier zreq_tom_fic;
};

struct ZREQ_TOM {
  char zreq_tom_name[4];          /* Monitor name          */
  char zreq_tom_func[1];          /* Function type         */
  char zreq_tom_tabn[1];          /* Request type           */
  char zreq_tom_reqn[8];          /* Request number         */
  char zreq_tom_rtcf[1];          /* Tom return code        */
  char zreq_tom_rscf[3];          /* Reason return code      */
  union uni_sci uni;
};
```

d1b8ruex.h File

The d1b8ruex.h file includes a header with a description of the Pi99 used with partners of type TOM (st_pi99). This structure is only valid between Connect:Express monitors. If the partner type is Other, the Pi99 field is available. The following structure (d1b8ruex) is used for PeSIT and FTP transfers.

```
/*************************************************************************/
/*                                                                       */
/*                     D1B8RUEX.H FILE                                   */
/*                                                                       */
/*************************************************************************/

struct st_pi99 {
       /*--- VERSION IDENTIFIER ---*/
       char pesit_version[1];
       char pi99_version[1];
       /*--- REQUEST ELEMENTS ---*/
       char request_number[4];
       char request_date[4];
       char performance_flag[1];
       char type[1];
       char group12[2];
       /*--- RACF ELEMENTS ---*/
       char user_remote_id[8];
       char old_password[8];
       char new_password[8];
       /*--- ELEMENTS OF SENDER FILE ---*/
       char s_dsname[44];
       char s_blksize[2];
       char s_lrecl[2];
       char s_recfm[1];
       char s_dcb_format[1];
       char s_disp1[1];
       char s_unit[8];
       char s_volcount[1];
       char s_volser1[6];
       char s_volser2[6];
       char s_volser3[6];
       char s_volser4[6];
       char s_volser5[6];
       char group21[1];
       char group31[1];
       char tape_only[6];
       char group42[2];
       /*--- ELEMENT OF RECEIVER FILE ---*/
       char r_dsname[44];
       char r_blksize[2];
       char r_nbdir[2];
       char transmit_retpd[2];
       char r_lrecl[2];
       char r_recfm[1];
       char r_dcb_format[1];
       char r_disp1[1];
       char r_disp2[1];
       char r_disp3[1];
       char r_recfm[1];
       char r_dcb_format[1];
       char r_disp1[1];
       char r_disp2[1];
       char r_disp3[1];
```

Continued

```
        char expdt[3];
        char r_unit[8];
        char r_volcount[1];
        char r_volser1[6];
        char r_volser2[6];
        char r_volser3[6];
        char r_volser4[6];
        char r_volser5[6];
        char device_dependent_seg[14];
        char group52[2];
        char last_byte[1];
};      /* 254 taille max */
/*********************************************************************/
/*                                                                   */
/*  Temporary file structure given during exit execution.            */
/*                                                                   */
/*********************************************************************/
struct d1b8ruex {
  char subsys[4];       /* Monitor name                              */
  char date_launch[14];
  char request[8];      /* Local Request Number                      */
  char file[8];         /* Symbolic filename                         */
  char partner[8];      /* Symbolic partner name                     */
  char dsn[44];         /* Dsname                                    */
  char direction[1];    /* Transfer direction                        */
  char type[1];         /* Exit type 'I'(nit), 'E'(nd) or 'F'(ailed) */
  char trfid[8];        /* Identification of transfer                */
  char local[8];        /* Local symbolic name                       */
  char src[4];          /* System Return Code                        */
  char trc[4];          /* Tom Return Code                           */
  char diag[4];         /* Protocol Return Code                      */
  char requester[8];    /* Requester Symbolic Name                   */
  char server[8];       /* Server Symbolic Name                      */
  char tyf[5];          /* File Type                                 */
  char priority[1];     /* Priority                                  */
  char cod[1];          /* Data Code (0: Ascii, 1: Ebcdic, 2: Binary)*/
  char label[80];       /* File Label                                */
  char dhc[12];         /* File Creation Date and Time               */
  char dhd[12];         /* File Extraction Date and Time             */

  union {
    struct st_pi99 pi99_new;
    char pi99_old[64];
  } pi99;               /* Pi 99 (Depends on Protocol Version)       */
                        /* #1 : First 64 bytes with 'TOM' in EBCDIC  */
                        /* #2 : 1st Byte : 0X02 2nd Byte : 0X01      */
  char begining[14];    /* Beginning of transfer date/time           */
  char ending[14];      /* Ending of transfer date/time              */
  char duration[8];     /* Transfer duration. Unused                 */
  char kbytes[7];       /* Kbytes Transfered                         */
  char records[10];     /* Records Transfered                        */
  char lrecl[5];        /* Record Length                             */
  char recfm[2];        /* Record Format (TV, TF, BF, BU)            */
  char kbytes_lfs[12];/* LARGE FILE SUPPORT                        */
  char s_pi99_254[254]; /* Sender pi99 on 254                        */
  char r_pi99_254[254]; /* Receiver pi99 on 254                      */
  char user_org[8];     /* User Origin                               */
  char user_dst[8];     /* User Destination                          */
  char user_snd[24];    /* User Sender pi61                          */
 char user_rcv[24];    /* User receiver pi62                        */
 };
```

The following structure (d1b8etb3) is passed to the ETEBAC3 exit.

```
/*********************************************************************/
/*                                                                   */
/*  Temporary file structure given during ETEBAC3 exit execution.    */
/*                                                                   */
/*********************************************************************/
struct d1b8etb3 {
  char client_or_bank[1]; /*   0 */  /* 'C' = client, 'B' = Bank */
  char msg_type[3];       /*   1 */  /* 'DEB' 'FIN' */
  char msg_orig[1];       /*   4 */  /* T=set by TOM C=by Client B=by Bank*/
  char param_card[80];    /*   5 */
  char dial_number[15];   /*  85 */
  char user_data[8];      /* 100 */
  char oknok[20];         /* 108 */
  char user_retcode[4];   /* 128 */
  char tom_retcode[4];    /* 132 */
  char request[8];        /* 138 */
  char physical_name[44]; /* 144 */
  char client[8];         /* 188 */
  char bank[8];           /* 196 */
  char filler[52];        /* 204 */
};                        /* size = 256 */
```

Appendix B

Compilation Procedures

This appendix provides examples of the compilation procedure and the Make executable.

Compilation Procedure

The compilation of a user program that calls the function L0B8Z20 must include the option -L $TOM_DIR/itom. The environment variable corresponds to the root directory of the monitor, for example /home/tom1.

The following example shows a procedure for the compilation of the program.

```
# This procedure enables you to compile a source file in the OBJECT module.
echo ""
echo "Compilation of the Source Program test_api.c"
echo ""
make -f makefile.test_api
if [ $? = "0" ]
then echo "No error....."
else
     echo "Error....."
     exit 1
fi
```

Make Executable Procedure

The following example shows a procedure for the Make executable.

```
# make file for test_api

# For the end
fin        : test_api
     @ls -l *.lis >> trace.lis
     @cat trace.lis

test_api   : test_api.o
         cc -g -o test_api test_api.o -L $(TOM_DIR)/itom -lc -litom \
         2> ld_test_api.lis

test_api.o :  test_api.c
         cc -c test_api.c 2>cc_test_api.lis
```