CIW
Approved Pages
Automatic Data Conversion
by Jeff Hildebrand

Ballot to CP

Page 11: change 3rd paragraph " • **Mapped** conversation ....."

Mapped convesations allow programs to exchange arbitrary **data records** in data formats agreed upon by the application programmers. CPI Communications supports both the traditional pass through method, where the application program encodes and decodes the data, and external routines that encode and decode user data.

Page 22: add to advanced function calls Send_Mapped_Data, Receive_Mapped_Data, Set_Mapped_Initialization_Data, Extract_Mapped_Initialization_Data.

Page 34: add after "Automatic Conversion Characteristics" section

Automatic Data Conversion

Automatic data conversion allows two mapped conversation programs to exchange data records in a manner that insulates a program from concerns about how the partner program views the data record. This is accomplished by encoding and decoding data records using routines that are external to the user application. These routines can be written by a systems administrator, some other qualified group, or a vendor. The MAP_NAME is used to access the correct routine. Within the SNA architecture this would be a separate routine which the underlying layers would access. A particular implementation may either have the user link edit the routine into the application program or use dynamic link libraries, to the end user the results are the same. Within the OSI architecture this name would instruct the U-ASE (the OSI TP entity responsible for encoding and decoding user data) what specific U-ASE ASN.1 structure to use. When a data record is received the reverse process occurs with the decoder recognizing what it received and generating the appropriate local MAP_NAME.

When an application program is moving data between homogeneous systems and intends to perform all manipulation of the data within the application program itself, it may use the network like a pipe and simply send and receive buffers of data. The program can also accomplish this by using pure Send_Data and Receive calls. It can also accomplish this by using Send_Mapped_data and Receive_Mapped_Data calls and not supplying a map name or supplying one with all blanks. The SNA routine is

the vendor installed routine which supports CPI Communications version 1.0 mapped conversations. The OSI routine is the Unstructured Data Transfer U-ASE (UDT).

The MAP_NAME is a, 1 - 64 character, local name defined by the person who wrote the encode/decode routine. Both sides of the conversation could have different local names for the same mapping function. This might occur when a transaction traveled across an organizational boundary where the naming could not be coordinated. A good example would be two companies which use an industry standard data record and have independently created a MAP_NAME for the data record. SNA maps the local MAP_NAME to a globally unique network MAP_NAME. The globally unique name is provided by a responsible systems administrator. OSI uses the application context to define, among other things, the abstract syntax's (ASN.1 encodings) the conversation can use. The MAP_NAME is a local name established by a systems administrator for the U-ASE to identify what encoding to use. Again, the two different systems may have different MAP_NAMES. Unlike SNA, OSI does not transmit a MAP_NAME across the network. Instead the self defining nature of ASN.1 automatically performs this function.

The encode/decode routines need not be mirror images of each other. The ability to manipulate the user data before it is sent or received by the application program allows the application programmer to be insulated from the network. The receiver may only require a subset of the data that was sent, so only specific fields are delivered to the program. This allows changes in the data a program sends to occur without requiring every receiving program to change, only the decoding routines need change. The sending program may only send a subset of the required data with the encoding routine filling in the missing fields.

Automatic data conversion assume that both partners behave correctly. There may be cases where Send_Data and Receive calls are mixed with Send_Mapped_Data and Receive_Mapped_Data. These calls can be mixed, if the following assumptions are met:

1. Both partners use equivalent map routines.
2. The Send_Data partner's map routine can recognize the incoming buffer and encode correctly. For LU 6.2, the map routine will generate the correct map name and send it to the partner.
3. The Receive_Data partner application code is able to recognize the decoded buffer and act accordingly.

The send/receive mapped data routines all use a Map_Name_Length of 0 to 64. A zero length indicates a NULL map name and the use of either the LU 6.2 supplied map routine or the OSI UDT ASE, depending on the underlying CRM.

Page 36: add to Initialization queue: Set_Mapped_Initialization_Data and Extract_Mapped_Initialization_Data.

Page 36: add to Send queue: Send_Mapped_Data.

Page 36: add to Receive queue: Receive_Mapped_Data.

Page 38 - Table 7 add

Receive_Mapped_Data
Send_Mapped_Data

Page 42: change in Data Conversion, second paragraph, last sentence to: These calls may be used to write a program which is independent of partner program's local character set.

Page 42: add after last paragraph of Data Conversion.

With the Send_Mapped_Data, Receive_Mapped_Data, Set_Mapped_Initialization_Data, Extract_Mapped_Initialization_Data calls, CPI Communications provides access to sophisticated routines for conversion of data in the user buffer, before it is sent or received. These routines may be used to write programs that are independent of the encoding of the partner program. This allows the encode/decode routine to change the order of the fields, selectively receive fields, and supply missing fields when sending.

These calls should not be used when using the OSI Unstructured Data Transfer U-ASE or CPI Communications version 1.0 mapped conversations. If the user desires to use these calls for uniformity with other programs the MAP_NAME should be set to Blanks and/or MAP_NAME_LENGTH set to zero.

Page 102: add to call reference table.

Send_Mapped_Data CMSNDM Used by a program to send mapped data to its partner.

Receive_Mapped_Data CMRCVM Used by a program to receive mapped data to its partner.

Set_Mapped_Initialization_Data CMSMID Used by a program to set the initialization_data conversation characteristic for mapped initialization data.

Extract_Mapped_Initialization_Data CMEMID Used by a program to extract the initialization_data conversation characteristic for mapped initialization data.

Page 111: add to 2nd paragraph list.

Call CMSMID - Set_Mapped_Initialization_Data

Page 114: add 8a (renumber list)

Initialization data specified by the use of the Set_Mapped_Initialization_Data (CMSMID) call is sent to the remote program along with the conversation startup request. The remote program may extract the initialization data with the Extract_Mapped_Initialization_Data (CMEMID).

Page 115: add to related information.

Set_Mapped_Initialization_Data CMSMID on page X. Used by a program to set the initialization_data conversation characteristic for mapped initialization data.

Page 119: add item 5, after (CMSID).

or Set_Mapped_Initialization_Data (CMSMID)

Page 119: add to related Information.

Set_Mapped_Initialization_Data CMSMID on page X. Used by a program to set the initialization_data conversation characteristic for mapped initialization data.

Extract_Mapped_Initialization_Data CMEMID on page X. Used by a program to extract the initialization_data conversation characteristic for mapped initialization data.

Page 127 :add to related information.

Send_Mapped_Data CMSNDM on page X. Used by a program to send mapped data to its partner.

Receive_Mapped_Data CMRCVM on page X. Used by a program to receive mapped data to its partner.

Page 129 :add to related information.

Send_Mapped_Data CMSNDM on page X. Used by a program to send mapped data to its partner.

Receive_Mapped_Data CMRCVM on page X. Used by a program to receive mapped data to its partner.

Page 155: add to related information.

Set_Mapped_Initialization_Data CMSMID on page X. Used by a program to set the initialization_data conversation characteristic for mapped initialization data.

Extract_Mapped_Initialization_Data CMEMID on page X. Used by a program to extract the initialization_data conversation characteristic for mapped initialization data.

Page 179: add to related information.

Send_Mapped_Data CMSNDM on page X. Used by a program to send mapped data to its partner.

Page 206 :add to related information.

Send_Mapped_Data CMSNDM on page X. Used by a program to send mapped data to its partner.

Receive_Mapped_Data CMRCVM on page X. Used by a program to receive mapped data to its partner.

Page 217 :add to related information.

Receive_Mapped_Data CMRCVM on page X. Used by a program to receive mapped data to its partner.

Page 223 :add to related information.

Send_Mapped_Data CMSNDM on page X. Used by a program to send mapped data to its partner.

Receive_Mapped_Data CMRCVM on page X. Used by a program to receive mapped data to its partner.

Page 267 :add to related information.

Set_Mapped_Initialization_Data CMSMID on page X. Used by a program to set the initialization_data conversation characteristic for mapped initialization data.

Extract_Mapped_Initialization_Data CMEMID on page X. Used by a program to extract the initialization_data conversation characteristic for mapped initialization data.

Page 292 :add to related information.

Receive_Mapped_Data CMRCVM on page X. Used by a program to receive mapped data to its partner.

Page 298 :add to related information.

Send_Mapped_Data CMSNDM on page X. Used by a program to send mapped data to its partner.

Page 318 -table 16 (Variable characteristics)

Add to call_id section

CM_CMEMID	76
CM_CMSMID	77
CM_CMSNDM	78
CM_CMRCVM	79

Add to return code section

CM_UNKNOWN_MAP_NAME_REQUESTED	50
CM_UNKNOWN_MAP_NAME_RECEIVED	51
CM_MAP_ROUTINE_ERROR	52

Page 329 Table 18 variable types: add

map_name	Character string	T61String	0 - 64 bytes
map_name_length	Integer	N/A	32 bits

Page 337 Appendix B return codes - add:

CM_UNKNOWN_MAP_NAME_REQUESTED

This return code is returned on Send_Mapped_Data, or Set_Mapped_Initialization_Data when a MAP_NAME is specified that the underlying map routine finds is invalid. The function requested is not performed and the application program can perform appropriate error recovery. A user should check to ensure that the proper map routine is being used and the spelling of the MAP_NAME.

CM_UNKNOWN_MAP_NAME_RECEIVED

This return code is returned on Receive_Mapped_Data, or Extract_Mapped_Initialization_Data when the incoming map name is invalid because the map routine cannot recognize it. The function request returns the un-decoded incoming data in the buffer and expects the application program to take appropriate action. This is a serious error which indicates that the two partners do not have the necessary common understanding to communicate. Under the OSI TP CRM this would indicate one of the partners has violated the association's application context. Under the LU 6.2 CRM both partners should review what is expected and determine which partner is in error.

CM_MAP_ROUTINE_ERROR

This return code is returned on Send_Mapped_Data, Receive_Mapped_Data, Extract_Mapped_Initialization_Data, and Set_Mapped_Initialization_Data indicating the underlying map routine aborted. If the error occurred during a receive function the incoming data is placed in the buffer un-decoded so the application program can take appropriate action.

Page 354 After table 20 (no secondary return codes): add

CM_UNKNOWN_MAP_NAME_REQUESTED
CM_UNKNOWN_MAP_NAME_RECEIVED
CM_MAP_ROUTINE_ERROR

Page 356 - table 22 add (Secondary return codes)

209 The send length, calculated by the map routine, specifies a value less than 1 or more than 86.

210 The conversation_type is set to CM_BASIC_CONVERSATION.

Page 372: add to Return Code values: mp mp means one of the following:

- CM_UNKNOWN_MAP_NAME_REQUESTED
- CM_UNKNOWN_MAP_NAME_RECEIVED
- CM_MAP_ROUTINE_ERROR

Page 372: add to Return Code values: bs bs means

- CM_BUFFER_TOO_SMALL

Page 372: remove CM_BUFFER_TOO_SMALL from Return Code value: ed

Page 377: add 'bs' to Extract_Init_Data

Page 377: change 'ed' to 'bs' in Extract_partner_ID

Page 377: add duplicate existing Extract_Init_Data entry: rename Extract_Mapped_Initialization_Data and add to line 1 [ok,pc,mp,bs]

Page 378: add 'bs' to Rcv_Exp_Data

Page 378: add duplicate existing Receive(I) and Receive(W) table entries: rename to Receive_Mapped_Data(I)/(W) and add to first line [ok,mp]{dr, no}

Page 379: add duplicate existing send_data table entry: rename Send_mapped_data and add:

new line 12 [mp] - for states 3, 5, 11,12,13

Page 380: add duplicate existing Set_Initialization_Data entry: rename Set_Mapped_Initialization_Data and add to line 1 [ok,pc,mp]

Page 386: add to Return Code values: mp mp means one of the following:

- CM_UNKNOWN_MAP_NAME_REQUESTED
- CM_UNKNOWN_MAP_NAME_RECEIVED
- CM_MAP_ROUTINE_ERROR

Page 386: add to Return Code values: bs bs means

- CM_BUFFER_TOO_SMALL

Page 391: add 'bs' to Extract_Init_Data

Page 391: change 'ed' to 'bs' in Extract_partner_ID

Page 392: add 'bs' to Receive_Exp_Data

Page 392: add duplicate existing Receive(I) and Receive(W) table entries: rename to Receive_Mapped_Data(I)/(W) and add to first line [ok, mp] {dr, no}

Page 392: add duplicate existing send_data table entry: rename Send_mapped_data and add to line 8 [ae, mp]

Page 393: add duplicate existing Set_Initialization_Data entry: rename Set_Mapped_Initialization_Data and add to line 1 [ok,pc,mp]

Page 391: add duplicate existing Extract_Init_Data entry: rename Extract_Mapped_Initialization_Data and add to line 1 [ok,pc,mp,bs]

Page 398 change paragraph 7 to
CPI Communications does not support the USER_CONTROL_DATA function available with the LU 6.2 interface.

Page 398 eliminate first bullet under CPI Communications does not support the specifcation of MAP_NAME.

Page 401: add to LU 6.2 CRM mapping entries: duplicate calls non-mapped entry, but X in map_name.

Page 440 - Application Generator Pseudonym File

1. Page440 - Call ID

   MOVE 76 TO CM_CMEMID;

   MOVE 77 TO CM_CMSMID;

   MOVE 78 TO CM_CMSNDM;

   MOVE 79 TO CM_CMRCVM;

2. Add page 442 (return code values)

   MOVE 50 TO CM_UNKNOWN_MAP_NAME_REQUESTED

   MOVE 51 TO CM_UNKNOWN_MAP_NAME_RECEIVED

   MOVE 52 TO CM_MAP_ROUTINE_ERROR

3. Add to Page 445 CSP Initialization of the constant data

```
:recditem  name  = CM_CMEMID
           level = 10          occurs = 00001
           scope = GLOBAL

:recditem  name  = CM_CMSMID
           level = 10          occurs = 00001
           scope = GLOBAL

:recditem  name  = CM_CMSNDM
           level = 10          occurs = 00001
           scope = GLOBAL

:recditem  name  = CM_CMRCVM
           level = 10          occurs = 00001
           scope = GLOBAL

:recditem  name  = CM_UNKNOWN_MAP_NAME_REQUESTED
           level = 10          occurs = 00001
           scope = GLOBAL

:recditem  name  = CM_UNKNOWN_MAP_NAME_RECEIVED
           level = 10          occurs = 00001
           scope = GLOBAL

:recditem  name  = CM_EXPEDITED_MAPPED_DATA_AVAILABLE
           level = 10          occurs = 00001
           scope = GLOBAL


:recditem  name  = CM_MAP_ROUTINE_ERROR
           level = 10          occurs = 00001
           scope = GLOBAL
```

```
:recditem  name   = MAP_NAME
           level  = 77          occurs = 00001
           scope  = GLOBAL

:recditem  name   = MAP_NAME_LENGTH
           level  = 77          occurs = 00001
           scope  = GLOBAL


:item      name   = CM_CMEMID
           date   = '090994'    time   = '12:00:00'   type   = BIN
           bytes  = 0004        decimals       = 00   evensql       = N
   :eitem.

:item      name   = CM_CMSMID
           date   = '090994'    time   = '12:00:00'   type   = BIN
           bytes  = 0004        decimals       = 00   evensql       = N
   :eitem.

:item      name   = CM_CMSNDM
           date   = '090994'    time   = '12:00:00'   type   = BIN
           bytes  = 0004        decimals       = 00   evensql       = N
   :eitem.

:item      name   = CM_CMRCVM
           date   = '090994'    time   = '12:00:00'   type   = BIN
           bytes  = 0004        decimals       = 00   evensql       = N
   :eitem.


:item      name   = CM_UNKNOWN_MAP_NAME_REQUESTED
           date   = '090994'    time   = '12:00:00'   type   = BIN
           bytes  = 0004        decimals       = 00   evensql       = N
   :eitem.

:item      name   = CM_UNKNOWN_MAP_NAME_RECEIVED
           date   = '090994'    time   = '12:00:00'   type   = BIN
           bytes  = 0004        decimals       = 00   evensql       = N
   :eitem.

:item      name   = CM_EXPEDITED_MAPPED_DATA_AVAILABLE
           date   = '090994'    time   = '12:00:00'   type   = BIN
           bytes  = 0004        decimals       = 00   evensql       = N
   :eitem.


:item      name   = CM_MAP_ROUTINE_ERROR
           date   = '090994'    time   = '12:00:00'   type   = BIN
           bytes  = 0004        decimals       = 00   evensql       = N
   :eitem.
```

```
:item       name   = MAP_NAME
            date   = '090994'     time    = '12:00:00'    type    = CHA
            bytes  = 0004         decimals        = 00    evensql         = N
    :eitem.
:item       name   = MAP_NAME_LENGTH
            date   = '090994'     time    = '12:00:00'    type    = CHA
            bytes  = 0004         decimals        = 00    evensql         = N
    :eitem.
```

Page 480 - CPIC.H file

1. Add page 481

```
#define cmemid  CMEMID
#define cmsmid  CMSMID
#define cmsndm  CMSNDM
#define cmrcvm   CMRCVM

( under Base CPC-C constant & type defs)
typedef char MAP_NAME [64];
typedef CM_INT32 MAP_NAME_LENGTH;
```

2. Add page 483 /* call ID values */

```
#define CM_CMEMID    (CM_CALL_ID) 76
#define CM_CMSMID    (CM_CALL_ID) 77
#define CM_CMSNDM   (CM_CALL_ID) 78
#define CM_CMRCVM   (CM_CALL_ID) 79
```

3. Add page 487 (return code values)

```
#define
      CM_UNKNOWN_MAP_NAME_REQUESTED  (CM_RETURN_CODE) 50
#define
       CM_UNKNOWN_MAP_NAME_RECEIVED   (CM_RETURN_CODE) 51
#define
       CM_MAP_ROUTINE_ERROR                    (CM_RETURN_CODE) 52
```

5. Add page 489

```
CM_ENTRY cmemid (   unsigned char  CM_PTR,      /* conversation ID*/
                    unsigned char CM_PTR,       /* local map name */
                    CM_MAP_NAME_LENGTH CM_PTR,
                                                /* map name length*/
                    unsigned char  CM_PTR,      /* initialization data*/
                    CM_INT32 CM_PTR,            /*requested length*/
                    CM_INT32 CM_PTR,            /*initial data length*/
                    CM_RETURN_CODE CM_PTR);
                                                /* return code*/

CM_ENTRY cmsmid (   unsigned char  CM_PTR,      /* conversation ID*/
                    unsigned char CM_PTR,       /* local map name */
                    CM_MAP_NAME_LENGTH CM_PTR,
                                                /* map name length*/
                    unsigned char  CM_PTR,      /* initialization data*/
                    CM_INT32 CM_PTR,            /*initial data length*/
                    CM_RETURN_CODE CM_PTR);
                                                /* return code*/


CM_ENTRY cmrcvm (   unsigned char  CM_PTR,      /* conversation ID*/
                    unsigned char CM_PTR,       /* local map name */
                    CM_MAP_NAME_LENGTH CM_PTR,
                                                /* map name length*/
                    unsigned char  CM_PTR,      /*buffer*/
                    CM_INT32 CM_PTR,            /*requested length*/
                    CM_INT32 CM_PTR,            /*data received*/
                    CM_STATUS_RECEIVED CM_PTR
                                                /*status received*/
                    CM_CONTROL_INFROMATION_RECEIVED
                                                        _CM_PTR
                                /* control_received Information*/
                    CM_RETURN_CODE CM_PTR);
                                                /* return code*/


CM_ENTRY cmsndm (   unsigned char  CM_PTR,      /* conversation ID*/
                    unsigned char CM_PTR,       /* local map name */
                    CM_MAP_NAME_LENGTH CM_PTR,
                                                /* map name length*/
                    unsigned char  CM_PTR,      /*buffer*/
                    CM_INT32 CM_PTR,            /*send length*/
                    CM_CONTROL_INFROMATION_RECEIVED
                                                        _CM_PTR
```

```
                                        /* control_received Information*/
                                CM_RETURN_CODE CM_PTR);
                                                        /* return code*/
```

Page 480 add under C macros:

```
#define Receive_Mapped_Data                 cmrcvm
#define Send_Mapped_Data                    cmsndx
#define Extract_Mapped_Initialization_Data  cmemid
#define Set_Mapped_Initialization_Data      cmsmid
```

Page 494 COBOL Pseudonym File

1. Add page 494 under 01 CALL-ID

```
        88 CM_CMEMID          VALUE   76.
        88 CM_CMSMID          VALUE   77.
        88 CM_CMSNDM          VALUE   78.
        88 CM_CMRCVM          VALUE   79.
```

2. Add to page 496 under 01 CM-RETCODE

```
        88 CM_UNKNOWN_MAP_NAME_REQUESTED     VALUE 50.
        88 CM_UNKNOWN_MAP_NAME_RECEIVED      VALUE 51.
        88 CM_MAP_ROUTINE_ERROR              VALUE 52.
```

3. Add to page 498

```
        *
        * Mapped Conversation values
        01  MAP-NAME                 PIC X(64).
        *    1 -64 BYTES
        01  MAP-NAME-LENGTH          PIC 9(9) COMP-4.
```

Add Page 501 FORTRAN Pseudonym File

1. Add page 501 under CALL-ID

```
        INTEGER   CM_CMEMID         /76/
```

```
INTEGER   CM_CMSMID       /77/
INTEGER   CM_CMSNDM       /78/
INTEGER   CM_CMRCVM       /79/
```

2. Add page 503 under CALL-ID

```
INTEGER   EMID        /76/
INTEGER   SMID        /77/
INTEGER   SNDM        /78/
INTEGER   RCVM        /79/
```

3. Add to page 507 under RETURN_CODE

```
INTEGER CM_UNKNOWN_MAP_NAME_REQUESTED    /50/
INTEGER CM_UNKNOWN_MAP_NAME_RECEIVED     /51/
INTEGER CM_MAP_ROUTINE_ERROR             /52/
INTEGER UMNR                             /50/
INTEGER UMNR                             /51/
INTEGER MRE                              /52/
```

4. Add to page 510 under ROUTINE PROTOTYPES

```
EXTERNAL    CM_CMEMID
EXTERNAL    CM_CMSMID
EXTERNAL    CM_CMSNDM
EXTERNAL    CM_CMRCVM
```

Page 512 PL/1 Pseudonym File

1. Add page 512 under CALL-ID

```
cm_cmemid       INITIAL(76),
cm_cmsmid       INITIAL(77),
cm_cmsndm       INITIAL(78),
cm_cmrcvm       INITIAL(79),
```

2. Add to page 513 under parameters

```
/*****************************************/
/*   Mapped Conversation values          */
```

```
/*****************************************/
DECLARE
    (cm_map_name              INITIAL(0),
     cm_map_name_length       INITIAL(1))
                              FIXED BINARY(31) STATIC;
```

3. Add to page 517 under Return codes

```
cm_unknown_map_name_requested     INITIAL(50),
cm_unknown_map_name_received      INITIAL(51),
cm_map_routine_error              INITIAL(52),
```

4. Add page 522 CPIC calls

```
DECLARE
CMEMID ENTRY (      CHAR(8),            /* In - conversation ID*/
                    CHAR(*),            /* Out - local map name */
                    FIXED BINARY(31),       /* Out - map name
    length*/
                    CHAR(*),            /* Out - initialization data*/
                    FIXED BINARY(31),       /* In - requested
    length*/
                    FIXED BINARY(31),       /* Out - initial data
    length*/
                    FIXED BINARY(31))/* Out - return code */
                EXTERNAL OPTIONS(ASM, INTER);


DECLARE
CMSMID ENTRY (      CHAR(8),            /* In - conversation ID*/
                    CHAR(64),           /* In -local map name */
                    FIXED BINARY(31) /* In - map name length*/
                    CHAR(*),            /* In - initialization data*/
                    FIXED BINARY(31),       /* In - initial data
    length*/
                    FIXED BINARY(31)) /* Out - return code */
                EXTERNAL OPTIONS(ASM, INTER);


DECLARE
CMRCVM ENTRY(CHAR(8),               /* In -conversation ID*/
                CHAR(*),            /* Out - local map name */
                FIXED BINARY(31),           /* Out - map name
    length*/
                CHAR(*),            /* Out - buffer*/
```

```
                FIXED BINARY(31),           /*In - requested
    length*/
                FIXED BINARY(31),           /*Out - data
    received*/
                FIXED BINARY(31),           /*Out - status
    received*/
                FIXED BINARY(31),           /* Out - control
    Information*/
                FIXED BINARY(31)) /* Out - Return code */
          EXTERNAL OPTIONS(ASM, INTER);


     CMSNDM ENTRY (CHAR(8),          /*In - conversation ID*/
                CHAR(64),           /* In - local map name */
                FIXED BINARY(31),       /* In -map name length*/
                CHAR(*),            /*In -buffer*/
                FIXED BINARY(31),       /*In - send length*/
                FIXED BINARY(31) /* Out - control Information*/
                FIXED BINARY(31)) /* Out - Return code */
          EXTERNAL OPTIONS(ASM, INTER);
```

Page 533 - REXX Pseudonym File

1. Page 533 - Call ID

   CM_CMEMID = 76

   CM_CMSMID = 77

   CM_CMSNDM = 78

   CM_CMRCVM = 79

2. Add page 535 (return code values)

   CM_UNKNOWN_MAP_NAME_REQUESTED = 50

   CM_UNKNOWN_MAP_NAME_RECEIVED = 51

   CM_MAP_ROUTINE_ERROR = 52

Page 538 - RPG Pseudonym File

1. Page538 - Call ID

   I* CM_CMEMID --VALUE 76 (EMID)

   I* CM_CMSMID --VALUE 77 (SMID)

   I* CM_CMSNDM --VALUE 78 (SNDM)

   I* CM_CMRCVM --VALUE 79 (RCVM)

I 76 C EMID

I 77 C SMID

I 78 C SNDM

I 79 C RCVM

2. Add page 544 (return code values)

I* CM_UNKNOWN_MAP_NAME_REQUESTED -- VALUE 50 (UMNR)

I* CM_UNKNOWN_MAP_NAME_RECEIVED -- VALUE 51 (UMNR)

I* CM_MAP_ROUTINE_ERROR -- VALUE 52 (MRE)

I 50 C UMNR

I 51 C UMNR

I 52 C MRE

LU 6.2 Mappings

Add to page 401 - Table 32 (Relationship of LU 6.2 verbs to CPIC)

Put all these entries under advanced functions

Duplicate RECEIVE line:

Change name to Receive_Mapped_Data and put X in the Map_Name column under MC_RECEIVE_DATA.. Leave all colums under RECEIVE blank.

Duplicate Send line:

Change name to Send_Mapped_Data and put X in the Map_Name column under MC_SEND_DATA.. Leave all colums under SEND blank.

Duplicate Set_Initialization_Data line:

Change name to Set_Mapped_Initialization_Data. The Map name is not supported for MC_ALLOCATE so the MAP NAME is only used locally to prepare PIP data. Leave all columns under ALLOCATE blank.

Add row for Extract_Mapped_Initialization_Data. This call is for OSI TP so leave all columns blank.

OSI TP Mappings

Page 407 Mapping CPI-C to OSI TP Services - add papargraph 2

Initialization data is used through out this section, the handling of this data should be done in the following manner:

> If the application is using the Unstructured_Data_Transfer ASE it should use Set_initilization_Data and Extract_Initialization_Data function calls.
>
> If the application is **NOT** using the Unstructured_Data_Transfer ASE it should use Set_Mapped_initilization_Data and Extract_Mapped_Initialization_Data function calls.

Add to page 408

CMRCVM - Receive_Mapped_Data (See Table 50Aon page 415 for details.)	TP-BEGIN-DIALOGUE response TP-BEGIN-TRANSACTION request TP-GRANT-CONTROL request
CMSNDM- Send_Mapped_Data (See Table 51A on page 415 for details.)	TP-BEGIN-DIALOGUE response TP-BEGIN-TRANSACTION request (TP-DATA request) U-ASE TP-HANDSHAKE request TP-GRANT-CONTROL request TP-HANDSHAKE-AND-GRANT-CONTROL request TP-DEFERRED-GRANT-CONTROL request TP-END-DIALOGUE request TP-DEFERRED-END-DIALOGUE request TP-U-ABORT request

Add to page 415

Table 50A. CMRCVM - RECEIVE_MAPPED_DATA

CPI-C Call Parameters and Conversation Characteristics	OSI TP Service Parameters
if: Confirmation = true on TP-BEGIN-DIALOGUE indication **and** it is the first activity on the conversation following Accept_Conversation or Accept_Incoming	
CMRCVM - RECEIVE_MAPPED_DATA = accepted *initialization_data* (characteristic)	TP-BEGIN-DIALOGUE response Result User-Data
if: *conversation_state* = CM_SEND_STATE and *sync_level* = CM_SYNC_POINT or *sync_level* = CM_SYNC_POINT_NO_CONFIRM and *transaction_control* = CM_UNCHAINED_TRANSACTIONS **and** *begin_transaction* = CM_BEGIN_IMPLICIT **and** it is the first activity on the conversation following the start of the current transaction	
CMRCVM - RECEIVE_MAPPED_DATA	TP-BEGIN-TRANSACTION request (no parameters) TP-GRANT-CONTROL request (no parameters)
if: *sync_level* = CM_SYNC_POINT or *sync_level* = CM_SYNC_POINT_NO_CONFIRM **and** *transaction_control* = CM_UNCHAINED_TRANSACTIONS **and** *begin transaction* = CM_BEGIN_IMPLICIT **and** it is the first activity on the conversation following the start of the current transaction	
CMRCVM - RECEIVE_MAPPED_DATA	TP-BEGIN-TRANSACTION request (no parameters)
if: *conversation_state* = CM_SEND_STATE	
CMRCVM - RECEIVE_MAPPED_DATA	TP-GRANT-CONTROL request (no parameters)

Add to page 415

Table 51A (Page 1 of 5). CMSNDM - Send_Mapped_Data

CPI-C Call Parameters and Conversation Characteristics	OSI TP Service Parameters
if: *send type* = CM_BUFFER_DATA	

CMSNDM - Send__Mapped_Data	(See note at end of table.) (TP-DATA request) U-ASE
map_name (parameter)	(local U-ASE map routine identifier)
buffer (parameter)	User-Data

if: *send type* = CM_SEND_AND_FLUSH

CMSNDM - Send_Mapped_Data	(See note at end of table.) (TP-DATA request) U-ASE
map_name (parameter)	(local U-ASE map routine identifier)
buffer (parameter)	User-Data

if: *send_type* = CM_SEND_AND_CONFIRM

CPl-C Call Parameters and Conversation Characteristics	**OSI TP Service Parameters**
CMSNDM - Send_Mapped_Data	(See note at end of table.) (TP-DATA request) U-ASE
map_name (parameter)	(local U-ASE map routine identifier)
buffer (parameter)	User-Data TP-HANDSHAKE request
confrmation_urgency (characteristic)	Confirmation-Urgency

if: *send_type* = CM_SEND_AND_PREP_TO_RECEIVE
and *prepare_to_receive_type* = CMPREP_TO_RECEIVE_SYNC_LEVEL
and *sync_level* = CM_NONE

CMSNDM - Send_Mapped_Data	(See note at end of table.) (TP-DATA request) U-ASE
map_name (parameter)	(local U-ASE map routine identifier)
buffer (parameter)	User-Data TP-GRANT-CONTROL request (no parameters)

if: *send type* = CM_SEND_AND_PREP_TO_RECEIVE
and *prepare_to_receive_type* = CMPREP_TO_RECEIVE_SYNC_LEVEL
and *sync_level* = CM_CONFIRM

CMSNDM - Send_Mapped_Data	(See note at end of table.) (TP-DATA request) U-ASE
map_name (parameter)	(local U-ASE map routine identifier)
buffer (parameter)	User-Data TP-HANDSHAKE-AND-GRANT-CONTROL request
confirmation_urgency (characteristic)	Confirmation-Urgency

if: *send_type* = CM_SEND_AND_PREP_TO_RECEIVE
and *prepare_to_receive_type* = CMPREP_TO_RECEIVE_SYNC_LEVEL
and *sync_level* = CM_SYNC_POINT or *sync_level* = CM_SYNC_POINT_NO_CONFIRM
and *transaction_control* = CM_UNCHAINED_TRANSACTIONS
and *begin_transaction* = CM_BEGIN_IMPLICIT
and it is the first activity on the conversation followIng the start of the current transaction

CMSEND - Send_Data	(See note at end of table.) TP-BEGIN-TRANSACTION request (no parameters) (TP-DATA request) U-ASE
buffer (parameter)	User-Data TP-DEFERRED-GRANT-CONTROL request (no parameters)

if: *send_type* = CM_SEND_AND_PREP_TO_RECEIVE
and *prepare_to_receive_type* = CM_PREP_TO_RECEIVE_SYNC_LEVEL
and *sync_level* = CM_SYNC_POINT or *sync_level* = CM_SYNC_POINT_NO_CONFIRM
and *(transaction_control* = CM_CHAINED_TRANSACTIONS or *(transaction_control* = CM_UNCHAINED_TRANSACTIONS and the conversation is included in the current transaction))

CMSNDM - Send_Mapped_Data	(See note at end of table.) (TP-DATA request) U-ASE
map_name (parameter)	(local U-ASE map routine identifier)
buffer (parameter)	User-Data TP-DEFERRED-GRANT-CONTROL request (no parameters)

if: *send_type* = CM_SEND_AND_PREP_TO_RECEIVE
and *prepare_to_receive_type* = CM_PREP_TO_RECEIVESYNC_LEVEL
and *sync_level* = CM_SYNC_POINT
and *(transaction_control* = CM_UNCHAINED_TRANSACTIONS and the conversation is not currently included in the transaction)

CMSNDM - Send_Mapped_Data	(See note at end of table.) (TP-DATA request) U-ASE
map_name (parameter)	(local U-ASE map routine identifier)
buffer (parameter)	User-Data TP-HANDSHAKE-AND-GRANT-CONTROL request
confirmation_urgency (characteristic)	Confirmation-Urgency TP-GRANT-CONTROL request (no parameters)

if: *send_type* = CM_SEND_AND_PREP_TO_RECEIVE
and *prepare_to_receive_type* = CM_PREP_TO_RECEIVE_SYNC_LEVEL
and *sync_level* = CM_SYNC_POINT_NO_CONFIRM

and *(transaction_control* = CM_UNCHAINED_TRANSACTIONS and the conversation is not currently included in the transaction)

CMSNDM - Send_Mapped_Data	(See note at end of table.) (TP-DATA request) U-ASE
map_name (parameter)	(local U-ASE map routine identifier)
buffer (parameter)	User-Data

if: *send_type* = CM_SEND_AND_PREP_TO_RECEIVE
and *prepare_to_receive_type* = CM_PREP_TO_RECEIVE_FLUSH

CMSNDM - Send_Mapped_Data	(See note at end of table.) (TP-DATA request) U-ASE
map_name (parameter)	(local U-ASE map routine identifier)
buffer (parameter)	User-Data TP-GRANT-CONTROL request (no parameters)

if: *send_type* = CM_SEND_AND_PREP_TO_RECEIVE
and *prepare_to_receive_type* = CM_PREP_TO_RECEIVE_CONFIRM

CMSNDM - Send_Mapped_Data	(See note at end of table.) (TP-DATA request) U-ASE
map_name (parameter)	(local U-ASE map routine identifier)
buffer (parameter)	User-Data TP-HANDSHAKE-AND-GRANT-CONTROL request
confirmation_urgency (characteristic)	Confirmation-Urgency

if: *send_type* = CM_SEND_AND_DEALLOCATE
and *deallocate_type* = CM_DEALLOCATE_SYNC_LEVEL
and *sync_level* = CM_NONE

CMSNDM - Send_Mapped_Data	(See note at end of table.) (TP-DATA request) U-ASE
map_name (parameter)	(local U-ASE map routine identifier)
buffer (parameter)	User-Data TP-END-DIALOGUE request
=false	Confirmation

if: *send_type* = CM_SEND_AND_DEALLOCATE
and *deailocate_type* = CM_DEALLOCATE_SYNC_LEVEL
and *sync_level* = CM_CONFIRM

CMSNDM - Send_Mapped_Data	(See note at end of table.) (TP-DATA request) U-ASE
map_name (parameter)	(local U-ASE map routine identifier)
buffer (parameter)	User-Data TP-END-DIALOGUE request
=true	Confirmation

if: *send_type* = CM SEND-AND_DEALLOCATE
and *deallocate_type* = CM_DEALLOCATE_SYNC_LEVEL
and *sync_level* = CM_SYNCPOINT or *sync_level* = CM_SYNCPOINT_NO_CONFIRM
and *transaction_control* = CM_UNCHAINED_TRANSACTIONS
and *begin_transachon* = CM_BEGIN_IMPLICIT
and it is the first activity on the conversation following the start of the current transaction

CMSNDM - Send_Mapped_Data	(See note at end of table.) (TP-DATA request) U-ASE
map_name (parameter)	(local U-ASE map routine identifier)
buffer (parameter)	User-Data TP_DEFERRED_END_DIALOGUE request (no parameters)

if: *send_type* = CM_SEND_AND_DEALLOCATE
and *deallocate_type* = *CM_DEALLOCATE_SYNC_LEVEL*
and *Sync_level* = CM_SYNC_POINT or *Sync_Ievel* = CM_SYNC_POINT_NO_CONFIRM
and *(transaction_control* = CM_CHAINED_TRANSACTIONS or *(transaction_control* = CM_UNCHAINED_TRANSACTIONS and the conversation is included in the current transaction))

CMSNDM - Send__Mapped_Data	(See note at end of table.) (TP-DATA request) U-ASE
map_name (parameter)	(local U-ASE map routine identifier)
buffer (parameter)	User-Data TP-DEFERRED-END-DIALOGUE request (no Parameters)

if*: send_type* = CM_SEND_AND_DEALLOCATE
and *deallocatetype* = CM_DEALLOCATE_SYNC_LEVEL
and *sync_ievel* = CM_SYNC_POINT
and *(transact,oncontrol* = CM_UNCHAINED_TRANSACTIONS and the conversation is not currently included in the transaction)

CMSNDM - Send__Mapped_Data	(See note at end of table.) (TP-DATA request) U-ASE
map_name (parameter)	(local U-ASE map routine identifier)
buffer (parameter)	User-Data TP-END-DIALOGUE request
true	Confirmation

if: *send_type* = CM_SEND_AND_DEALLOCATE
and *deallocate_type* = CM_DEALLOCATE_SYNC_LEVEL
and *sync_level* = CM_SYNC_POINT_NO_CONFIRM
and *(transaction_control* = CM_UNCHAINED_TRANSACTIONS and the conversation is not currently included in the transaction)

CMSNDM - Send_Mapped_Data	(See note at end of table.) (TP-DATA request) U-ASE
map_name (parameter)	(local U-ASE map routine identifier)
buffer (parameter)	User-Data
false	TP-END-DIALOGUE request Confirmation

if: *send_type* = CM_SEND_AND_DEALLOCATE
and *deallocate_type* = CM_DEALLOCATE_FLUSH

CMSNDM - Send_Mapped_Data	(See note at end of table.) (TP-DATA request) U-ASE
map_name (parameter)	(local U-ASE map routine identifier)
buffer (parameter)	User-Data
=false	TP-END-DIALOGUE request Confirmation

if: *send_type* = CM_SEND_AND_DEALLOCATE
and *deallocate_type* = CM_DEALLOCATE_CONFIRM

CMSNDM - Send_Mapped_Data	(See note at end of table.) (TP-DATA request) U-ASE
map_name (parameter)	(local U-ASE map routine identifier)
buffer (parameter)	User-Data
=true	TP-END-DIALOGUE request Confirmation

if: *send_type* = CM_SEND_AND_DEALLOCATE
and *deallocate_type* = CM_DEALLOCATE_ABEND

CMSNDM - Send_Mapped_Data	(See note at end of table.) (TP-DATA request) U-ASE
map_name (parameter)	(local U-ASE map routine identifier)
buffer (parameter)	User-Data
log_data (characteristic)	TP-U-ABORT request User-Data

Note: if Confirmation = true on TP-BEGIN-DIALOGUE indication
and it is the first activity on the conversation following Accept_Conversation or Accept_Incoming
then the identified OSI TP Service primitive(s) will be preceded by TP-BEGIN-DIALOGUE response (Result = accepted and User-Data = *initialization_data* (characteristic)).

Add and Change page 419

(TP-DATA indication) U-ASE indication	Completes an outstanding CMRCVM - Receive_Mapped_Data and sets *map_name*
TP-BEGIN-DIALOGUE confirm (See Table 55 on page 421 for details.)	Completes an outstanding CMRCV - Receive or CMRCVM - Receive_Mapped_Data and maps to *control_information_received* values.
TP-END-DIALOGUE indication (See Table 56 on page 422 for details.)	Completes an outstanding CMRCV. Receive or CMRCVM - Receive_Mapped_Data and maps to status_received or *return_code* values.
TP-END-DIALOGUE confirm (no parameters)	Completes an outstanding CMSEND- Send_Data or CMSNDM Send_Mapped_Data CMDEAL - Deallocate and maps to *return_code* = CM_OK
TP-U-ERROR indication no parameters) Receive_Mapped_Data	Completes an outstanding CMRCV - Receive or CMRCVM - CMDEAL - Deallocate and maps to *return_code* = CM_PROGRAM_ERROR_PURGING.
TP-U-ABORT indication (See Table 57 on page 422 for details.)	Completes an outstanding CMRCV -Receive or CMRCVM - Receive_Mapped_Data and maps to *return_code* values.
TP-P-ABORT indication See Table 58 on page 423 for details.) Receive_Mapped_Data	Completes an outstanding CMRCV - Receive or CMRCVM - and maps to return_code values.
TP-GRANT.CONTROL indication (no parameters) Receive_Mapped_Data	Completes an outstanding CMRCV - Receive or CMRCVM - and maps to *status_received* = CM_SEND_RECEIVED
TP-REQUEST-CONTROL indication (no parameters)	Maps to *control_information_received* = CM_REQ_TO_SEND_RECEIVED

TPHANDSHAKE indication (no parameters) Receive_Mapped_Data	Completes an outstanding CMRCV - Receive or CMRCVM - and maps to *status_received* CM_CONFIRM_RECEIVED
TP-HANDSHAKE confirm (nc parameters)	Completes an outstanding CMCFM - Confirm CMSEND- Send_Data or CMSNDM Send_Mapped_Data and maps to *return_code=CM_OK.*

Add or Change page 420

TP-HANDSHAKE-AND-GRANT-CONTROL indication (no parameters)	completes an outstanding CMRCV Receive or CMRCVM - Receive_Mapped_Data and maps to *status_received* = CM_CONFIRM_SEND_RECEIVED
TP-HANDSHAKE-AND-GRANT-CONTROL confirm (no parameters)	Completes an outstanding CMPTR PrepareTo_Receive CMSEND- Send_Data or CMSNDM Send_Mapped_Data and maps to *return code=CM* OK
TP-BEGlN.TRANSACTION indication (no parameters)	Completes an outstanding CMRCV - Receive or CMRCVM - Receive_Mapped_Data and maps to *status_received* CM_JOIN_TRANSACTION.
TP-PREPARE indication (See Table 59 on page 423 for details.)	Completes an outstanding CMRCV Receive or CMRCVM - Receive_Mapped_Data and maps to *status_received* values

Add to page 425

CMRCVM - Receive_Mapped_Data (See Table 68A *on* page 429 for details.)	TP-BEGlN-DIALOGUE response TP-BEGlN-TRANSACTION request
CMSNDM - Send_Mapped_Data (See Table 69A *on* page 429 for details.)	TP-BEGlN-DlAL0GUE response TP-BEGlN-TRANSACTION request (TP-DATA request) UD-TRANSFER request

TP-END-DIALOGUE request
TP-DEFERRED-END-DIALOGUE request
TP-U-ABORT request

Add to Page 429

Table 68A CMRCVM - Receive_Mapped_Data

CPI-C Call Parameters and Conversation Characteristics	**OSI TP Service Parameters**
if: Confirmation = true on TP-BEGIN-DIAL0GUE indication and it is the first activity on the conversation following Accept_Conversation or Accept_Incoming	
CMRCVM - Receive_Mapped_Data	TP-BEGIN-DIAL0GUE response
= accepted	Result
initialization_data (characteristic)	User-Data
if: *sync_level* = CM_SYNC_POINT_NO_CONFIRM **and** *transaction_control* = CM_UNCHAINED_TRANSACTIONS and *begin_transaction* = CM_BEGIN_IMPLICIT **and** it is the first activity on the conversation following the start of the current transaction	
CMRCVM - Receive_Mapped_Data	TP-BEGIN-TRANSACTI0N request (no parameters)

Table 69A CMSNDM - Send_Mapped_Data

CPI-C Call Parameters and Conversation Characteristics	**OSI TP Service Parameters**
if: *send_type* = **CMBUFFER_DATA**	
	(See note at end of table.)
CMSNDM Send_Mapped_Data	(TP-DATA request) U-ASE request
buffer (parameter)	User-Data
map_name (parameter)	local U-ASE routine map routine identifier
if: *send_type* = **GM-SEND_AND_FLUSH**	
	(See note at end of table.)
CMSNDM Send_Mapped_Data	(TP-DATA request) U-ASE request
buffer (parameter)	User-Data
map_name (parameter)	local U-ASE routine map routine identifier
if: *send_type* = **CMSEND_AND_DEALLOCATE**	

and *deallocate_type* = **GM_DEALLOCATE_SYNC_LEVEL**
and *sync_level*= **GM_NONE**

	See note at end of table.)
CMSNDM Send_Mapped_Data	(TP-DATA request) U-ASE request
buffer (parameter)	User-Data
map_name (parameter)	local U-ASE routine map routine identifier
	TP-END-DIALOGUE request
false	Confirmation
=false	(and TP-END-DIALOGUE indication Confirmation if (TP-END-DIALOGUE or 7P-oABORT indication) has not been processed)

if: *send_type* = **CMSEND_AND_DEALLOGATE**
and *deallocate_type* = **GM_DEALLOCATE_SYNC_LEVEL**
and *sync_level* = **GM_SYNC_POINT_NO_CONFIRM**
and *transaction_control* = **GM_UNGHAINED_TRANSACTIONS**
and *(begin_transacton* = **GM_BEGIN_IMPLICIT**
and It is the first activity on the conversation following the start of the current transaction)

CMSNDM Send_Mapped_Data	(See note at end of table.)
	TP-BEGIN-TRANSACTION request (no parameters)
	(TP-DATA request) U-ASE request
buffer (parameter)	User-Data
map_name (parameter)	local U-ASE routine map routine identifier
	TP-DEFERRED-END-DIALOGUE request

if: send_type = **CM_SEND AND_DEALLOCATE**
and *deallocate_type* = **CMDEALLOCATE_SYNC_LEVEL**
and *sync_level* = **CM_SYNC_POINT NO_CONFIRM**
and *((transaction_control* = **CM_UNCHAINED_TRANSACTIONS**
and not *(begin_transacton* = **CM_BEGIN_IMPLICIT**
and it is the first activity on the conversation following the start of the current transaction)) or
transaction_control = **CM_CHAINED_TRANSACTIONS)**
and the conversation is included in a transaction

	(See note at end of table.)
CMSNDM Send_Mapped_Data	(TP-DATA request) U-ASE request
buffer (parameter)	User-Data
map_name (parameter)	local U-ASE routine map routine identifier
	TP-DEFERRED-END-DIALOGUE request (no parameters)

if: *send_iype* = **CM_SEND_AND .DEALLOCATE**
and *deallocate_type* = **CMDEAriLOCATE_SYNC_LEVEL**
and *sync_level* = **CM SYNC POINT NO_CONFIRM**
and *((transaction_control* = **CM_UNCHAINED_TRANSACTIONS**
and not *(begin_transaction* = **CM_BEGIN_IMPLICIT**
and it is the first activity on the conversation following the start of the current transaction)) **or**
transaction_control = **CM_CHAINED_TRANSACTIONS)**
and the conversation is **not** currently included in a transaction

	(See note at end of table.)
CMSNDM Send_Mapped_Data	(TP-DATA request) U-ASE request
buffer (parameter)	User-Data
map_name (parameter)	local U-ASE routine map routine identifier
	TP-END-DIALOGUE request
false	Confirmation
	(and TP-END-DIALOGUE indication
false	Confirmation
	if (TP-END-DIALOGUE or TP-o-ABORT indication) has not been processed)

if: *send_type* **= CM_SEND AND_DEAULOCATE**
and *deallocate_type* = **CMDEALLOCATE_ FLUSH**

	(See note at end of table.)
CMSNDM Send_Mapped_Data	(TP-DATA request) U-ASE request
buffer (parameter)	User-Data
map_name (parameter)	local U-ASE routine map routine identifier
	TP-END-DIALOGUE request
= false	Confirmation
	(and TP-END-DIALOGUE indication
= false	Confirmation
	if (TP-END-DIALOGUE or TP-ABORT indication) has not been processed)

if: *send_type* = **CM_SEND AND DEALLOCATE**
and *deallocate_type* = **CM_DEALLOCATE_CONFIRM**

	(See note at end of table.)
CMSNDM Send_Mapped_Data	(TP-DATA request) U-ASE request
buffer (parameter)	User-Data
map_name (parameter)	local U-ASE routine map routine identifier
	if no collision with CM_ERROR_* or (ok.(*,jt)) **or** (ok.(o,cd))
	TP-END.DIALOGUE request
= true	Confirmation
	other wise return. respectively. CM_DEALLOC_CONFlRM_REJECT (S), **or** TP-U-ABORT request (Rollback **=** true). CM_DEALLOCATED_ABEND.begin_transaction_collisio n (RI and CM_DEALLOCATED_ABEND.begin_transaction-- collision (S). **or** TP-U-ABORT request (Rollback = false). CM_DEALLOCATED_ABEND dealloc_confirm_collision (R) and CM_DEALLOCATED_ABEND.dealloc_confirm_collision (S) If collision results in a return code to the Receive call. replace the *status_received* value that caused the collision

---

if: *send_type* = **CM_SEND_AND_DEALLOCATE**
and *deallocate_type* = **CM_DEALLOCATE_ABEND**

	(See note at end of table.)
CMSNDM Send_Mapped_Data	(TP-DATA request) U-ASE request
buffer (parameter)	User-Data
map_name (parameter)	local U-ASE routine map routine identifier
	TP-U-ABORT request
log_*data* (characteristic)	User-Data

Note: **if** confirmation **=** true on TP-BEGIN-DIALOGUE indication
and it is the first activity on the conversation following Accept_Conversation or Accept_Incoming
then the identified OSI TP Service primitive will be preceded by TP-BEGlN-DIALOGUE response (Result **=** accepted and User-Data
= *initalizaton_data* (characteristic))

Add to page 432

TP-BEGIN-DIALOGUE confirm (See Table 55 on page 421 for details.)	completes an outstanding CMRCV - Receive or CMRCVM- Receive_Mapped_Data and maps to *control_information_received* values
TF-END-DIALOGUE indication (See Table 56 on page 422 for details.)	Completes an outstanding CMRCV - Receive or CMRCVM- Receive_Mapped_Data and maps to status_received or return_code values
TP-END-DIALOGUE confirm (no parameters	Completes an outstanding CMSEND - Send_Data or CMSNDM - Send_Mapped_Data CMDEAL- Deallocate and maps to *return_code=CM_OK(S)* and **CM_DEALLOCATE_NORMAL** (R)
TP-U-ERROR indication (no parameters)	Completes an outstanding CMRCV- Receive or CMRCVM- Receive_Mapped_Data CMDEAL Deallocate(Confirm) and maps to *return_code* = CM_PROGRAM_ERROR_PURGING (R) (if CMDEAL - DEALLOCATE(Conf)rm) is being processed. also maps to Cll DEALLOC_CONFIRM_REJECT(S))
TP-U-ABORT indication (See Table 57 on page 422 for details.)	Completes an outstanding CMRCV Receive or CMRCVM- Receive_Mapped_Data and maps to *return_code* values
TP-P-ABORT indication (See Table 58 on page 423 for details)	Completes an outstanding CMRCV - Receive or CMRCVM- Receive_Mapped_Data CMDEAL - DEALLOCATE(Confirm) and maps to *return_code* values.

add to page 433

TP-BEGIN-TRANSACTION indication (no parameters)	**If** collision with outstanding CMDEAL (Conform), **then** Completes the outstanding CMDEAL Deallocate(Confirm) and maps to return_code = CM_DEALLOCATED_ABEND.begin_transaction_collision (R)and CM_DEALLOCATED_ABEND.begin_transaction_collision (S) and TP-U-ABORT request (Rollback = true) **else** if *conv_state* = SEND_RECEIVE **then** completes CMRCV- Receive or CMRCVM- Receive_Mapped_Data and maps to *status_received* = CM_JOIN_TRANSACTION
TF-PREPARE indication (See Table 53 on page 420 for details.)	Completes an outstanding CMRCV - Receive or CMRCVM- Receive_Mapped_Data and maps to status_received values

Receive_Mapped_Data (CMRCVM)

A program issues the Receive_Mapped_Data (CMRCVM) to receive

Before issuing the Receive_Mapped_Data call a program has the option of issuing the CALL CMSRT- Set_Receive_Type call.

```
CALL CMRCVM     (conversation_ID,
                 map_name,
                 map_name_length,
                 buffer,
                 requested_length,
                 data_received,
                 status_received,
                 control_information_received,
                 return_code)
```

Parameters

conversation_ID (input)

Specifies the variable containing the conversation identifier assigned to the conversation.

map_name (output)

Specifies the variable containing the mapping function used to decode the data record. The length of the variable must be at least 64 bytes.

map_name_length (output)

Specifies the variable containing the length of the returned map_name parameter.

buffer (output)

Specifies the variable in which the program is to receive the data.

Note: *Buffer* contains data only if *return_code* is set to CM_OK or CM_DEALLOCATE_NORMAL and *data_received* is not set to CM_NO_DATA_RECEIVED. If the *return_code* is set to CM_MAP_ROUTINE_ERROR or CM_UNKNOWN_MAP_NAME_RECEIVED the buffer contains the unprocessed user data..

requested_length (input)

Specifies the maximum amount of data the program is to receive. Valid *requested_length* values range from 0 to the maximum buffer size supported by

the system. and the map name. See usage note 10 on page xxx for additional information about determining the maximum buffer size.

data_received(output)

Specifies whether or not the program received data.

Note: Unless *return_code* is set to CM_OK or CM_DEALLOCATE_NORMAL, the value contained in *data_received* has no meaning.

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

- CM_NO_DATA_RECEIVED
  No data is received by the program. Status may be received if the *return_code* is set to CM_OK.
- CM_COMPLETE_DATA_RECEIVED
  A complete data record or the last remaining portion of the record is received.
- CM_INCOMPLETE_DATA_RECEIVED
  Less than a complete data record has been received.

received_length (output)

Specifies the variable containing the amount of data the program received, up to the maximum. If the program does not receive data on this call, the value contained in *received_length* has no meaning.

Note: Data is received only if *return_code* is set to CM_OK or CM_DEALLOCATE_NORMAL and *data_received* is not set to CM_NO_DATA_RECEIVED.

status_received (output)

Specifies the variable containing an indication of the conversation status.

Note: Unless *return_code* is set to CM_OK, the value contained in *status_received* has no meaning.

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

- CM_NO_STATUS_RECEIVED
  No conversation status is received by the program; data may be received..
- CM_SEND_RECEIVED (half-duplex conversations only)
  The remote program's end of the conversation in **Send-Pending** state (if the program also received data on this call) or **Send** state (if the program did not receive data on this call). The local program (which issued the Receive call) can now send data.
- CM_CONFIRM_RECEIVED (half-duplex conversations only)
  The remote program has sent a confirmation request, requesting the local program to respond by issuing a Confirmed call. The local program must respond by issuing Confirmed, Send_Error, or Deallocate with *deallocate_type* set to CM_DEALLOCATE_ABEND.
- CM_CONFIRM_SEND_RECEIVED (half-duplex conversations only)
  The remote program's end of the conversation has entered **RECEIVE** state with conformation requested. The local program must respond by issuing Confirmed, Send_Error, or Deallocate with *deallocate_type* set to

CM_DEALLOCATE_ABEND. Upon issuing a successful Confirmed call, the local program (which issued the Receive call) can now send data.

- CM_CONFIRM_DEALLOC_RECEIVED
  The remote program has deallocated the conversation with conformation requested. The local program must respond by issuing Confirmed, Send_Error, or Deallocate with *deallocate_type* set to CM_DEALLOCATE_ABEND. Upon issuing a successful Confirmed call, the local program (which issued the Receive call) is deallocated - that is, placed in **RESET** state.

For a conversation with *sync_level* set to CM_SYNC_POINT or CM_SYNC_POINT_NO_CONFIRM, and with the conversation include in a transaction, the *status_received variable* can also be set to one of the following values:

- CM_TAKE_COMMIT
  The remote program has issued a resource recovery commit call. The local program should issue a commit call in order to commit all protected program should issue a commit call in order to commit all protected resources throughout the transaction. When appropriate, the local program may respond by issuing a call other than commit, such as Send_error (for half-duplex conversations only) or a resource recovery backout call.
- CM_TAKE_COMMIT_SEND (half-duplex conversations only)
  The remote program has issued a Prepare_To_Receive call with *prepare_to_receive_type* set to CM_PREP_TO_RECEIVE_SYNC_LEVEL and *sync_level* set to CM_SYNC_POINT and then issued a resource recovery commit call. The local program should issue a commit call in order to commit all protected resources throughout the transaction. When appropriate, the local program may respond by issuing a call other than commit, such as Send_Error or a resource recovery backout call. If a successful commit call is issued, the local program can then send data.
- CM_TAKE_COMMIT_DEALLOCATE
  The remote program has deallocated the conversation with *deallocate_type* set to CM_DEALLOCATE_SYNC_LEVEL and *sync_level* set to CM_SYNC_POINT or CM_SYNC_POINT_NO_COMMIT and then issued a resource recovery commit call. The local program should issue a commit call in order to commit all protected resources throughout the transaction. The local program may respond by issuing a call other than commit, such as Send_Error for a half-duplex conversation, or a resource recovery backout call. If successful commit call is issued, the local program is then deallocated - that is placed in **RESET** state.
- CM_PREPARE_OK
  By issuing a Prepare (CMPREP) call, the local program requested that the remote program prepare its resources for commitment, and the remote program has done so, by issuing a commit call. The subtree is now ready to commit its resources.

For a conversation with *sync_level* set to CM_SYNC_POINT or CM_SYNC_POINT_NO_CONFIRM, and allocated using an OSI TP CRM, the *status_received variable* can also be set to one of the following values if the conversation is include in a transaction :

- CM_JOIN_TRANSACTION
  The remote program has requested that the local program join into its current transaction. The local program should issue a tx_begin call to the X/OPEN TX interface to join the superior's transaction as soon as any local work that is not to be included in the remote program's transaction has been completed.

- CM_TAKE_COMMIT_DATA_OK
  The remote program issued a Prepare call. For the exact conditions for receipt of this status_received value, refer to Table 15 and Table 16. The local program should issue a commit call in order to commit all protected resources throughout the transaction. The program is allowed to send data before issuing the commit call. When appropriate, the local program may respond by issuing a call other than commit, such as Send_Error (for half-duplex conversations only) or a resource recovery backout call.
- CM_TAKE_COMMIT_SEND_DATA_OK (half-duplex conversations only)
  The remote program issued a Prepare call. For the exact conditions for receipt of this status_received value, refer to Table 15 and Table 16. The local program should issue a commit call in order to commit all protected resources throughout the transaction. The program is allowed to send data before issuing the commit call. When appropriate, the local program may respond by issuing a call other than commit, such as Send_Error or a resource recovery backout call. If a successful commit call is issued, the local program can then send data.
- CM_TAKE_COMMIT_DEALLOC_DATA_OK
  The remote program issued a Prepare call. For the exact conditions for receipt of this status_received value, refer to Table 15 and Table 16. The local program should issue a commit call in order to commit all protected resources throughout the transaction. The program is allowed to send data before issuing the commit call. The local program may respond by issuing a call other than commit when appropriate, such as Send_Error for a half-duplex conversation, or a resource recovery backout call. If a successful commit call is issued, the local program is then deallocated--that is, placed in RESET state.

control_information_received (output)

Specifies the variable containing an indication of whether or not control information has been received.

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

- CM_NO_CONTROL_INFO_RECEIVED
  Indicates that no control information was received.
- CM_REQ_TO_SEND_RECEIVED (half-duplex conversations only)
  The local program received a request-to-send notification from the remote program. The remote program issued Request_To_Send, requesting the local program's end of conversation to enter **Receive** State, which would place the remote program's end of conversation in **Send** state. See "Request_To_Send(CMRTS)" on page 212 for further discussion of the local program's possible responses.
- CM_ALLOCATE_CONFIRM (OSI TP CRM only)
  The local program received confirmation of the remote program's acceptance of the conversation.
- CM_ALLOCATE_CONFIRM_WITH_DATA (OSI TP CRM only)
  The local program received confirmation of the remote program's acceptance of the conversation. The local program may now issue an Extract_Mapped_Initialization_Data (CMEMID) call to receive the initialization data.
- CM_ALLOCATE_REJECT_WITH_DATA (OSI TP CRM only)
  The remote program rejected the conversation. The local program may now

issue an Extract_Mapped_Initialization_Data (CMEMID) call to receive the initialization data.

This value will be returned with a return code of CM_OK. The program will receive a CM_DEALLOCATE_ABEND return code on a later call on the conversation.

- CM_EXPEDITED_DATA_AVAILABLE (LU 6.2 CRM only)
  Expedited data is available to be received.
- CM_EXPEDITED_MAPPED_DATA_AVAILABLE (LU 6.2 CRM only)
  Mapped expedited data is available to be received.
- CM_RTS_RCVD_AND_EXP_MAP_DATA_AVAIL (half-duplex conversations only and LU 6.2 CRM only)
  The local program received a request-to-send notification from the remote program and mapped expedited data is available to be received.

Notes:

1. If *return_code* is set to CM_PROGRAM_PARAMETER_CHECK or CM_PROGRAM_STATE_CHECK, the value contained in *control_information_received* has no meaning.
2. When more than one piece of control information is available to be returned tot he program, it will be returned in the following order:
   - CM_ALLOCATE_CONFIRMED,
     CM_ALLOCATE_CONFIRMED_WITH_DATA or
     CM_ALLOCATE_REJECTED_WITH_DATA
   - CM_RTS_RCVD_AND_EXP_DATA_AVAIL
   - CM_RTS_RCVD_AND_MAP_EXP_DATA_AVAIL
   - CM_REQ_TO_SEND_RECEIVED
   - CM_EXPEDITED_DATA_AVAILABLE
   - CM_NO_CONTROL_INFO_RECEIVED

return_code (output)

Specifies the result of call execution. The return_code that can be returned depend on the state and characteristics of the conversation at the time this call is issued.

The following return codes apply to half-duplex conversations:

If *receive_type* is set to CM_RECEIVE_AND_WAIT and this call is issued in **Send** state, return_code can have one of the following values:

- CM_OK
- CM_OPERATION_INCOMPLETE
- CM_CONVERSATION_TYPE_MISMATCH
- CM_PIP_NOT_SPECIFIED_CORRECTLY
- CM_SECURITY_NOT_VALID
- CM_SYNC_LVL_NOT_SUPPORTED_PGM
- CM_SYNC_LVL_NOT_SUPPORTED_SYS
- CM_TPN_NOT_RECOGNIZED
- CM_TP_NOT_AVAILABLE_NO_RETRY
- CM_TP_NOT_AVAILABLE_RETRY
- CM_DEALLOCATED_ABEND

- CM_DEALLOCATED_NORMAL
- CM_PROGRAM_ERROR_NO_TRUNC
- CM_PROGRAM_ERROR_PURGING
- CM_RESOURCE_FAILURE_NO_RETRY
- CM_RESOURCE_FAILURE_RETRY
- CM_OPERATION_NOT_ACCEPTED
- CM_UNKNOWN_MAP_NAME_RECEIVED
  The received data requires a map routine the local map function does not support. The buffer contains the unprocessed user data.
- CM_MAP_ROUTINE_ERROR
  The map routine encountered a problem with the received data. The buffer contains the unprocessed user data.
- CM_PRODUCT_SPECIFIC_ERROR
- CM_PROGRAM_STATE_CHECK
- The following values are returned only when the *sync_level is* set to CM_SYNC_POINT or CM_SYNC_POINT_NO_CONFIRM and the conversation is included in a transaction:
  - CM_TAKE_BACKOUT
  - CM_DEALLOCATED_ABEND_BO
  - CM_RESOURCE_FAIL_NO_RETRY_BO
  - CM_RESOURCE_FAILURE_RETRY_BO

If *receive_type* is set to CM_RECEIVE_AND_WAIT and this call is issued in **Send-Pending** state, return_code can have one of the following values:

- CM_OK
- CM_OPERATION_INCOMPLETE
- CM_SYNC_LVL_NOT_SUPPORTED_SYS
- CM_DEALLOCATED_ABEND
- CM_DEALLOCATED_NORMAL
- CM_PROGRAM_ERROR_NO_TRUNC
- CM_PROGRAM_ERROR_PURGING
- CM_RESOURCE_FAILURE_NO_RETRY
- CM_RESOURCE_FAILURE_RETRY
- CM_UNKNOWN_MAP_NAME_RECEIVED
  The received data requires a map routine the local map function does not support. The buffer contains the unprocessed user data.
- CM_MAP_ROUTINE_ERROR
  The map routine encountered a problem with the received data. The buffer contains the unprocessed user data.
- CM_OPERATION_NOT_ACCEPTED
- CM_PRODUCT_SPECIFIC_ERROR
- The following values are returned only when the *sync_level is* set to CM_SYNC_POINT or CM_SYNC_POINT_NO_CONFIRM and the conversation is included in a transaction:
  - CM_TAKE_BACKOUT

- CM_DEALLOCATED_ABEND_BO
- CM_RESOURCE_FAIL_NO_RETRY_BO
- CM_RESOURCE_FAILURE_RETRY_BO

If *receive_type* is set to CM_RECEIVE_AND_WAIT or CM_RECEIVE_IMMEDIATE and this call is issued in **Receive** state, return_code can have one of the following values:

- CM_OK
- CM_OPERATION_INCOMPLETE
  This value is only received when *receive_type* is set to CM_RECEIVE_AND_WAIT.
- CM_CONVERSATION_TYPE_MISMATCH
- CM_PIP_NOT_SPECIFIED_CORRECTLY
- CM_SECURITY_NOT_VALID
- CM_SYNC_LVL_NOT_SUPPORTED_PGM
- CM_SYNC_LVL_NOT_SUPPORTED_SYS
- CM_TPN_NOT_RECOGNIZED
- CM_TP_NOT_AVAILABLE_NO_RETRY
- CM_TP_NOT_AVAILABLE_RETRY
- CM_DEALLOCATED_ABEND
- CM_DEALLOCATED_NORMAL
- CM_PROGRAM_ERROR_NO_TRUNC
- CM_PROGRAM_ERROR_PURGING
- CM_RESOURCE_FAILURE_NO_RETRY
- CM_RESOURCE_FAILURE_RETRY
- CM_UNSUCCESSFUL
- CM_UNKNOWN_MAP_NAME_RECEIVED
  The received data requires a map routine the local map function does not support. The buffer contains the unprocessed user data.
- CM_MAP_ROUTINE_ERROR
  The map routine encountered a problem with the received data. The buffer contains the unprocessed user data.
- CM_OPERATION_NOT_ACCEPTED
- CM_PRODUCT_SPECIFIC_ERROR
- The following values are returned only when the *sync_level is* set to CM_SYNC_POINT or CM_SYNC_POINT_NO_CONFORM and the conversation is included in a transaction:
  - CM_TAKE_BACKOUT
  - CM_DEALLOCATED_ABEND_BO
  - CM_RESOURCE_FAIL_NO_RETRY_BO
  - CM_RESOURCE_FAILURE_RETRY_BO

If the state or parameter error has occurred, *return_code* can have one of the following values:

- CM_PROGRAM_STATE_CHECK
  This value indicates one of the following:

- The *receive_type* is set to CM_RECEIVE_AND_WAIT and the conversation is not set to CM_RECEIVE_AND_WAIT and the conversation is not in **Send, Send-Pending, Receive or Prepare** State.
- The *receive_type* is set to CM_RECEIVE_IMMEDIATE and the conversation is not set to CM_RECEIVE_AND_WAIT and the conversation is not in **Receive or Prepare** State.
- For a conversation with *sync_level* set to CM_SYNC_POINT or CM_SYNC_POINT_NO_CONFIRM, the conversation's context is in the **Backout-Required** condition. The Receive_Mapped_Data call is not allowed for this conversation while its context is in this condition.
- The program has received a *status_received* value of CM_JOIN_TRANSACTION and must issue a tx_begin call to the X/OPEN TX interface to join the transaction.

- CM_PROGRAM_PARAMETER_CHECK
  This value indicates one of the following:
  - The *conversation_ID* specifies an unassigned conversation identifier.
  - The requested_length specifies a value that exceeds the range permitted by the implementation. The maximum value of the length in each implementation is at least 32767. See usage note 11 on page xxx for additional information about determining the maximum buffer size.
  - The conversation_type characteristic is set to CM_BASIC_CONVERSATION.

The following return codes apply to full_duplex conversations:

If the call is issued in **Send-Receive** or **Receive-Only** state and either the *sync_level* is CM_NONE or the *sync_level* is CM_SYNC_POINT_NO_CONFIRM and the conversation is not currently included in a transaction, return_code can have one of the following values:

- CM_OK
- CM_CONVERSATION_TYPE_MISMATCH
- CM_PIP_NOT_SPECIFIED_CORRECTLY
- CM_SECURITY_NOT_VALID
- CM_SYNC_LVL_NOT_SUPPORTED_PGM
- CM_SYNC_LVL_NOT_SUPPORTED_SYS
- CM_SEND_RCV_MODE_NOT_SUPPORTED
- CM_TPN_NOT_RECOGNIZED
- CM_TP_NOT_AVAILABLE_NO_RETRY
- CM_TP_NOT_AVAILABLE_RETRY
- CM_DEALLOCATED_ABEND
- CM_DEALLOCATED_NORMAL
- CM_PROGRAM_ERROR_NO_TRUNC (half-duplex conversations only)
- CM_PROGRAM_ERROR_PURGING (OSI TP CRM only)
- CM_RESOURCE_FAILURE_NO_RETRY
- CM_RESOURCE_FAILURE_RETRY
- CM_OPERATION_NOT_ACCEPTED

- CM_UNKNOWN_MAP_NAME_RECEIVED
  The received data requires a map routine the local map function does not support. The buffer contains the unprocessed user data.
- CM_MAP_ROUTINE_ERROR
  The map routine encountered a problem with the received data. The buffer contains the unprocessed user data.
- CM_OPERATION_INCOMPLETE
- CM_PRODUCT_SPECIFIC_ERROR
- CM_PROGRAM_STATE_CHECK

The following clause are returned only if *sync_level* is CM_SYNC_POINT_NO_CONFIRM and the state is **Send-Receive or Prepared** and the conversation is included in a transaction:

- CM_TAKE_BACKOUT
- CM_DEALLOCATED_ABEND_BO
- CM_RESOURCE_FAILURE_NO_RETRY_BO
- CM_RESOURCE_FAILURE_RETRY_BO
- CM_CONV_DEALLOC_AFTER_SYNCPT
- CM_INCLUDE_PARTNER_REJECT_BO
- CM_PRODUCT_SPECIFIC_ERROR

If the state or parameter error has occurred, *return_code* can have one of the following values:

- CM_PROGRAM_STATE_CHECK
  This value indicates one of the following:
  - The conversation is not in **Send-Receive, Prepared,** or **Receive-Only** state.
  - For a conversation with *sync_level* set to CM_SYNC_POINT_NO_CONFIRM, the conversation's context is in the **Backout Required** condition. The Receive_Mapped_Data call in not allowed for this conversation while its context is in this condition.
- CM_PROGRAM_PARAMETER_CHECK
  This value indicates one of the following:
  - The *conversation_ID* specifies an unassigned conversation identifier.
  - The requested_length specifies a value that exceeds the range permitted by the implementation. The maximum value of the length in each implementation is at least 32767. See usage note 11 on page xxx for additional information about determining the maximum buffer size.
  - The conversation_type characteristic is set to CM_BASIC_CONVERSATION.

State Changes

For half-duplex conversations, when *return_code* indicates CM_OK:

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

— The *receive type* is set to CM_RECEIVE_AND_WAIT.

— The conversation is in **Send-Pending** or **Send** state.

— The *data_received* indicates CM_DATA_RECEIVED, CM_COMPLETE_DATA_RECEIVED or CM_INCOMPLETE_DATA_RECEIVED.

— The *status_received* indicates CM_NO_STATUS_RECEIVED.

- The conversation enters **Send** state when *data_received* is set to CM_NO_DATA_RECEIVED and *status_received* is set to CM_SEND_RECEIVED.

- The conversation enters **Send-Pending** state when *data_received* is set to CM_DATA_RECEIVED, or CM_COMPLETE_DATA_RECEIVED, and *status_received* is set to CM_SEND_RECEIVED.

- The conversation enters **Confirm, Confirm-Send, or Confirm-Deallocate** state when *status_received* is set to, respectively, CM_CONFIRM_RECEIVED, CM_CONFIRM_SEND_RECEIVED, or CM_CONFIRM_DEALLOC_RECEIVED.

- For a conversation with *sync_level* set to either CM_SYNC_POINT or CM_SYNC_POINT_NO_CONFIRM, the conversation enters Sync-Point, Sync-Point-Send, or Sync-Point-Deallocate state when *status_received* is set to CM_TAKE_COMMIT, CM_TAKE_COMMIT_SEND, or CM_TAKE_COMMIT_DEALLOCATE, respectively.

- For a conversation with *sync_level* set to either CM_SYNC_POINT or CM_SYNC_POINT_NO_CONFIRM, the conversation enters **Sync-Point, Sync-Point-Send**, or **Sync-Point-Deallocate** state when *status_received* is set to CM_TAKE_COMMIT_DATA_OK CM_TAKE_COMMIT_SEND_DATA_OK or CM_TAKE_COMMIT_DEALLOC_DATA_OK, respectively.
- No state change occurs when the call is issued in **Receive** state; *data_received* is set to CM_DATA_RECEIVED CM_COMPLETE_DATA_RECEIVED, or CM_INCOMPLETE_DATA_RECEIVED; and *status_received* indicates CM_NO_STATUS_RECEIVED.
- No state change occurs when the call is issued in **Prepared** state, or if *status_received* indicates CM_JOIN_TRANSACTION.

For full-duplex conversations, when *return_code* indicates CM_OK:

- No state change occurs when the call is issued in **Prepared** or **Receive-Only** state, or if *status_received* indicates CM_JOIN_TRANSACTION.
- The conversation enters **Confirm-Deallocate** state when *status_received* is set to CM_CONFIRM_DEALLOC_RECEIVED.
- For a conversation with *sync_level* set to CM_SYNC_POINT_NO_CONFIRM, the conversation enters Sync-Point state when *status_received* is set to CM_TAKE_COMMIT or CM_TAKE_COMMIT_DATA_OK, and it enters **Sync-Point-Deallocate** state when *status_received* is set to CM_TAKE_COMMIT_DEALLOCATE or CM_TAKE_COMMIT_DEALLOC_DATA_OK.

1. If *receive_type* is set to CM_RECEIVE_AND_WAIT and no information is present when the call is made, CPI Communications waits for information to arrive on the specified conversation before allowing the Receive call to return with the information. If information is already available, the program receives it without waiting.

2. For a half-duplex conversation, if the program issues a Receive call with its end of the conversation in **Send** state with *receive_type* set to CM_RECEIVE_AND_WAIT, the local system will flush its send buffer and send all buffered information to the remote program. The local system will also send a change-of-direction indication. This is a convenient method to change the direction of the conversation, because it leaves the local program's end of the conversation in Receive state and tells the remote program that it may now begin sending data. The local system waits for information to arrive.

Note: A Receive call in **Send** or **Send-Pending** state with a *receive_type* set to CM_RECEIVE_AND_WAIT generates an implicit execution of Prepare_To_Receive with *prepare_to_receive_type* set to CM_PREP_TO_RECEIVE_FLUSH followed by a Receive. Refer to “Prepare_To_Receive (CMPTR)” on page 188 for more information.

3. If *receive_type* is set to CM_RECEIVE_IMMEDIATE, a Receive call receives any available information, but does not wait for information to arrive. If information is available, it is returned to the program with an indication of the exact nature of the information received.

   Since data may not be available when a given Receive call is issued, a program that is using concurrent conversations with multiple partners might use a *receive_type* of CM_RECEIVE_IMMEDIATE and periodically check each conversation for data. For more information about multiple, concurrent conversations “Multiple Conversations” on page 23.

4. If the *return_code* indicates CM_PROGRAM_STATE_CHECK or CM_PROGRAM_PARAMETER_CHECK, the values of all other parameters on this call have no meaning.

5. A Receive call issued against a mapped conversation can receive only as much of the data record as specified by the *requested_length* parameter. The *data_received* parameter indicates whether the program has received a complete or incomplete data record, as follows:

- When the program receives a complete data record or the last remaining portion of a data record, the *data_received* parameter is set to CM_COMPLETE_DATA_RECEIVED. The length of the record or portion of the record is less than or equal to the length specified on the *requested_length* parameter.
- When the program receives a portion of the data record other than the last remaining portion, the *data_received* parameter is set to CM_INCOMPLETE_DATA_RECEIVED. The data record is incomplete for one of the following reasons

— *receive_type* is set to CM_RECEIVE_AND_WAIT. and the length of the record is greater than the length specified on the *requested_length* parameter.

— *receive_type* is set to CM_RECEIVE_IMMEDIATE, and either the length of the record is greater than the length specified on the *requested_length* parameter or the last portion of the data record has not arrived from the partner program.

In either case, the amount of data received is equal to the *received_length* specified.

6. The Receive call made with *requested_length* set to zero has no special significance. The type of information available is indicated by the *return_code, data_received,* and *status_received* parameters, as usual. If *receive_type* is set to CM_RECEIVE_AND_WAIT and no information is available, this call waits for information to arrive. If *receive type* is set to CM_RECEIVE_IMMEDIATE, - it is possible that no information is available. If data is available, the *data_received* parameter is set to CM_INCOMPLETE_DATA_RECEIVED, the program receives no data.

   If the conversation is mapped and a null data record is available (resulting from a Send_Mapped_Data call with *send_length* set to 0), the *data_received* parameter is set to CM_COMPLETE_DATA_RECEIVED and the *received_length* parameter is set to 0.

Note: When *requested length* is set to zero, receipt of either data or status can be indicated, but not both. The only exception to this rule is when a null data record is available for receipt on a mapped conversation. In that case, receipt of the null data record and status can both be indicated.

7. The program can receive both data and conversation status on the same call. However, if the remote program truncates a logical record, the local program receives the indication of the truncation on the Receive call issued by the local program after it receives all of the truncated record. The *return_code. data_received,* and *status_received* parameters indicate to the program the kind of information the program receives.
8. The program may receive data and conversation status on the same Receive call or on separate Receive calls. The program should be prepared for either case.
9. For a half-duplex conversation, the request-to-send notification is returned to the program in addition to (not in place of) the information indicated by the *return_code data_received,* and *status_received* parameters.
10. A program must not specify a value in the *requested_length* parameter that is greater than the maximum the implementation can support. The maximum may vary from system to system. The maximum also depends on the *map_name* parameter.

    Every CPI-C product must be able to receive data buffers with a length of 32,767 bytes. Users who write map routines should describe the length of the data buffer after decoding, if the encoded buffer has a length of 32,767 bytes.

    The application program should use a requested_length value not smaller that this value. The program should also be aware that CM_INCOMPLETE_DATA_RECEIVED value of the data_received parameter may be returned when the maximum buffer size differs across systems.
11. When the Receive call is processed in non-blocking mode and *receive type* is set to CM_RECEIVE_IMMEDIATE**,** the call completes immediately. If information is not available, *return_code* is set to CM_UNSUCCESSFUL.

12. When the local program has requested confirmation of the Allocate call and the first call made by the recipient program is Request_To_Send or the recipient program has issued a Send_Expedited_Data call. the CM_ALLOCATE_CONFIRMED value of the *control_information_received* parameter will be returned first and one of the following values will be returned at the next opportunity:

- CM_RTS_RCVD_AND_EXP_DATA_AVAIL
- CM_REQ_TO_SEND_RECEIVED
- CM_EXPEDITED_DATA_AVAILABLE
- CM_EXPEDITED_MAPPED_DATA_AVAILABLE

13. The Receive call may be issued following a successful Prepare call, without a state transition to **Receive** state in case of a hart-duplex conversation, for either of these reasons:
    - To receive data when CM_PREPARE_DATA_PERMITTED is selected and the Prepare Call is issued
    - To receive a new CM_PREPARE_OK value in *status_received.*
14. For a full-duplex conversation, if *receive type* is set to CM_RECEIVE_AND_WAIT and the conversation startup request has not been sent to the partner, then the Receive call will flush the conversation startup request to the partner.
15. For a full-duplex conversation, if the return code CM_PROGRAM_ERROR_PURGING is received, it indicates that the conversation is allocated using an OSI TP CRM and that data may have been purged. The application has to ensure that the two partners are coordinated.
16. For a full-duplex conversation, when CM_DEALLOCATED_ABEND or CM_DEALLOCATED_ABEND_BO is received, further information on the cause of the deallocation may be obtained by issuing the Extract_Secondary_Information call.
17. When *control_information_received* indicates that expedited data is available to be received, subsequent calls with this parameter will continue to indicate that expedited data is available until the expedited data has been received by the program.
18. The *received_length* is local information from the map routine. The received_length value is calculated by the map routine after all transformations are completed.
19. If a basic conversation data record is received it will be treated as an error by the map routine.

4.38.5 Related Information

Most of the example program flows in Chapter 3, “Program-to-Program Communication Tutorial” show programs using the Receive call.

4.41,“Request_To_Send (CMRTS)” on page 212 discusses how a program can place its end of the conversation into **Receive** state.

4.42.“Send_Mapped_Data (CMSNDM)” on page xxx provides more information on complete and incomplete logical records and data records.

4.68,“Set_Receive_Type (CMSRT)” on page 291 discusses the *receive_type* characteristic and its various values.

Send_Mapped_Data (CMSNDM)

A program uses the Send_Mapped_Data (CMSNDM) call to send data that requires some type of transformation, e.g. ASN.1 encoding, before it is sent to the remote partner. The *map_name* is used to identify to the local map routine what data is being sent so that it can apply the correct transforms.

When issued during a basic conversation, this call results in an error.

Before issuing the Send_Mapped_Data call, a program has the option of issuing one or more of the following calls, which affect the function of the Send_Data call:

CALL CMSST - Set_Send_Type

If ***send type =*** **CM_SEND_AND_PREP_TO_RECEIVE**, optional setup may include: **CALL CMSPTR - Set_Prepare_To_Receive_Type**

If *send type* = **CM_SEND_AND_DEALLOCATE,** optional setup may include: CALL CMSDT - Set_Deallocate_Type

Format

```
CALL CMSNDM     (conversation_ID,
                 map_name,
                 map_name_length,
                 buffer,
                 send_length,
                 control_information_received,
                 return_code)
```

Parameters

conversation_ID (input)
Specifies the conversation identifier of the conversation.

map_name (input)
Specifies the mapping function to be used to encode the data record. This is a 0 to 64 byte identifier.

map_name_length (input)
Specifies the length of the map_name. The length can be from 0 to 64 bytes. If zero, the *map_name* parameter is ignored.

buffer (input)
When a program issues a Send_Mapped_Data call *buffer* specifies the data record to be sent. The length of the data record is given by the *send_length* parameter.

send_length (input)
The *send_length* ranges in value from 0 to the maximum buffer size supported

by the system. The maximum buffer size depends on the map name. See Usage Note 6 on page 222 for additional information about determining the maximum buffer size. The *send_length* parameter specifies the size of the *buffer* parameter and the number of bytes to be sent to the local map routine. The map routine will calculate the length of the data to be sent on the conversation and handle it in accordance with underlying CRM.

When a program issues a Send_Mapped_Data call and *send_length* is zero, a null data record is sent the map routine will determine if a null record will be sent on the conversation. The user should refer to the site documentation for that particular map routine.

control_information_received (output)
Specifies the variable containing an indication of whether or not control information has been received.

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

- CM_NO_CONTROL_INFO_RECEIVED
  Indicates that no control information was received.
- CM_REQ_TO_SEND_RECEIVED (half-duplex conversations only) The local program received a request-to-send notification from the remote program. The remote program issued Request_To_Send, requesting the local program's end of the conversation to enter **Receive** state, which would place the remote program's end of the conversation in **Send** state. See "Request_To_Send (CMRTS)" on page 212 for further discussion of the local program's possible responses.
- CM_ALLOCATE_CONFIRMED (OSI TP CRM only) The local program received confirmation of the remote program's acceptance of the conversation.
- CM_ALLOCATE_CONFIRMED_WITH_DATA (OSI TP CRM only) The local program received confirmation of the remote program's acceptance of the conversation. The local program may now issue an Extract_Mapped_Initialization_Data (CMEMID) call to receive the initialization data.
- CM_ALLOCATE_REJECTED_WITH_DATA (OSI TP CRM only)

  The remote program rejected the conversation. The local program may now issue an Extract_Mapped_Initialization_Data (CMEMID) call to receive the initialization data.

  This value will be returned with a return code of **CM_OK.** The program will receive a CM_DEALLOCATED_ABEND return code on a later call on the conversation.
- CM_EXPEDITED_DATA_AVAILABLE (LU 6.2 CRM only) Expedited data is available to be received.
- CM_RTS_RCVD_AND_EXP_DATA_AVAIL (half-duplex conversations and LU 6.2 CRM only) The local program received a request-to-send notification from the remote program and expedited data is available to be received
- CM_EXPEDITED_MAPPED_DATA_AVAILABLE (LU 6.2 CRM only) Mapped expedited data is available to be received.
- CM_RTS_RCVD_AND_EXP_MAP_DATA_AVAIL (half-duplex conversations and LU 6.2 CRM only) The local program received a request-to-send notification from the remote program and expedited Mapped data is available to be received

Notes:

1. If *return_code* is set to CM_PROGRAM_PARAMETER_CHECK or CM_PROGRAM_STATE_CHECK, the value contained in *control_information_received* has no meaning.
2. When more than one piece of control information is available to be returned to the program, it will be returned in the following order:

- CM_ALLOCATE_CONFIRMED CM_ALLOCATE_CONFIRMED_WITH_DATA, or CM_ALLOCATE_REJECTED_WITH_DATA
- CM_RTS_RCVD_AND_EXP_DATA_AVAIL
- CM_REQ_TO_SEND_RECEIVED
- CM_EXPEDITED_DATA_AVAILABLE
- CM_EXPEDITED_MAPPED_DATA_AVAILABLE
- CM_NO_CONTROL_INFO_RECEIVED

return_code (output)

Specifies the result of the call execution. The *return_code* variable can have one of the following values.

The following return codes apply to half-duplex conversations:

- CM OK
- CM_OPERATION_INCOMPLETE
- CM_CONVERSATION_TYPE_MISMATCH
- CM_PIP_NOT_SPECIFIED_CORRECTLY
- CM_SECURITY_NOT_VALID
- CM_SYNC_LVL_NOT_SUPPORTED_PGM
- CM_SYNC_LVL_NOT_SUPPORTED_SYS
- CM_TPN_NOT_RECOGNIZED
- CM_TP_NOT_AVAILABLE_NO_RETRY
- CM_TP_NOT_AVAILABLE_RETRY
- CM_PROGRAM_ERROR_PURGING
- CM_DEALLOCATED_ABEND
- CM_RESOURCE_FAILURE_NO_RETRY
- CM_RESOURCE_FAILURE_RETRY
- CM_UNKNOWN_MAP_NAME_REQUESTED
  The supplied map name is unknown to the map routine, no data was sent.
- CM_MAP_ROUTINE_ERROR
  The map routine encountered a problem with the user data, no data was sent.

- CM_PROGRAM_STATE_CHECK

This value indicates one of the following:

— The conversation is not in **Send, Send-Pending, Sync-Point, Sync-Point-Send, or Sync-Point-Deallocate** state.

— The conversation is in **Sync-point, Sync-Point-Send**, **or Sync-Point-Deallocate** state, and the program receives a take-commit notification not ending in DATA_OK.

— For a conversation with *sync_level* set to CM_SYNC_POINT or CM_SYNC_POINT_NO_CONFIRM, the conversation's context is in the **Backout Required** condition. The Send_Mapped_Data call is not allowed for this conversation while its context is in this condition.

— The conversation is in **Sync-Point, Sync-Point-Send,** or **Sync-Point-Deallocate** state, and *send_type* is set to CM_SEND_AND_CONFIRM or CM_SEND_AND_PREP_TO_RECEIVE.

— The conversation is in **Sync-Point, Sync-Point-Send,** or **Sync-Point-Deallocate** state, the *send_type* is set to CM_SEND_AND_DEALLOCATE, and the *deallocate_type* is not set to CM_DEALLOCATE_ABEND.

— The *send_type* is set to CM_SEND_AND_DEALLOCATE and the following conditions are also true:

- The *deallocate_type* is set to CM_DEALLOCATE_FLUSH or CM_DEALLOCATE_CONFIRM.
- The *sync_level* is set to CM_SYNC_POINT or CM_SYNC_POINT_NO_CONFIRM,
- The conversation is included in a transaction.

- CM_PROGRAM_PARAMETER_CHECK

  This value indicates one of the following:

  — The *conversation_ID* specifies an unassigned conversation identifier.

  — The *send_length* exceeds the range permitted by the implementation.

  — The conversation_type characteristic is set to CM_BASIC_CONVERSATION.

  — The *send_type* is set to CM_SEND_AND_PREP_TO_RECEIVE and the following conditions are also true:

  The prepare_to_receive_type *is set to*

  CM_PREP_TO_RECEIVE_SYNC_LEVEL.

  - The *sync_level* is set to CM_SYNC_POINT or CM_SYNC_POINT_NO_CONFIRM.
  - The conversation is included in a transaction.
  - The conversation is using an OSI TP CRM, and the program is not the superior for the conversation.
  - The *send_type* is set to CM_SEND_AND_DEALLOCATE and the following conditions are also true:
  - The *deallocate_type* is set to CM_DEALLOCATE_SYNC_LEVEL.
  - The *sync_level* is set to CM_SYNC_POINT or CM_SYNC_POINT_NO_CONFIRM.
  - The conversation is included in a transaction.
  - The conversation is using an OSI TP CRM, and the program is not the superior for the conversation.

- CM_OPERATION_NOT_ACCEPTED
- CM_PRODUCT_SPECIFIC_ERROR

  The following values are returned only when *sync_level* is set to

CM_SYNC_POINT or CM_SYNC_POINT_NO_CONFIRM and the conversation is included in a transaction :

- CM_TAKE_BACKOUT
- CM_DEALLOCATED_ABEND_BO
- CM_RESOURCE_FAIL_NO_RETRY_BO
- CM_RESOURCE_FAILURE_RETRY_BO

The following return codes apply to full-duplex conversations.

The *return_code* can have one of the following values:

- CM OK
- CM_ALLOCATION_ERROR
- CM_DEALLOCATED_ABEND
- CM_DEALLOCATED_ABEND_SVC
- CM_DEALLOCATED_ABEND_TIMER
- CM_DEALLOCATED_CONFIRM_REJECT
- CM_RESOURCE_FAILURE_NO_RETRY
- CM_RESOURCE_FAILURE_RETRY
- CM_DEALLOCATED_NORMAL (OSI TP CRM only)
- CM_UNKNOWN_MAP_NAME_REQUESTED
  The supplied map name is unknown to the map routine, no data was sent.
- CM_MAP_ROUTINE_ERROR
  The map routine encountered a problem with the user data, no data was sent.

- CM_PROGRAM_STATE_CHECK

  This value indicates one of the following:

  — The conversation is not in **Send-Receive**, **Send-Only, Sync-Point,** or **Sync-Point-Deallocate** state.

  — The conversation is basic and in Send-receive or **Send-Only** state, the *send_type* is set to CM_SEND_AND_DEALLOCATE, the *deallocate_type* is not set to CM_DEALLOCATE_ABEND. and the data does not end on a logical record boundary.

  — The conversation is in **Sync-Point** or **Sync-Point Deallocate** state and the program received a take-commit notification not ending in

  DATA_OK.

  — The conversation is in **Sync-Point, or Sync-Point-Deallocate** state, the *send type* is set to CM_SEND_AND_DEALLOCATE, and the *deallocate_type* is not set to CM_DEALLOCATE_ABEND.

  — For a conversation with *sync_level* set to CM_SYNC_POINT_NO_CONFIRM, this return code indicates one of the following:

  — The conversation's context is in the **Backout-Required** condition. The Send_Data call is not allowed for this conversation while its context is in this condition.

— The local program has received a *status_received* value of **CM_JOIN_TRANSACTION** and must issue a tx_begin call to the X/Open TX interface to join the transaction.

- CM_PROGRAM_PARAMETER_CHECK

  This value indicates one of the following:

  — The *conversation_ID* specifies an unassigned conversation identifier.

  — The *send_length* exceeds the range permitted by the implementation. The maximum value of the length in each implementation is at least 32767.

  — The *send_type* is set to CM_SEND_AND_DEALLOCATE and the following conditions are also true:

  — The *deallocate_type* is set to CM_DEALLOCATE_FLUSH or CM_DEALLOCATE_CONFIRM.

  — The *sync_level* is set to CM_SYNC_POINT_NO_CONFIRM.

  — The conversation in included in a transaction.

  — The *send_type* is set to CM_SEND_AND_DEALLOCATE and the following conditions are also true:

  — The *deallocate_type* is set to CM_DEALLOCATE_SYNC_LEVEL.

  — The *sync_level* is set to CM_SYNC_POINT_NO_CONFIRM.

  — The conversation is included in a transaction.

  — The program is not the superior for the conversation.

  — The conversation_type characteristic is set to CM_BASIC_CONVERSATION.

- CM_OPERATION_NOT_ACCEPTED
- CM_OPERATION_INCOMPLETE
- CM_PRODUCT_SPECIFIC_ERROR

The following values are returned only if *sync_level* is CM_SYNC_POINT_NO_CONFIRM, the state is Send-Receive and the conversation is currently included in a transaction.

- CM_TAKE_BACKOUT
- CM_DEALLOCATED_ABEND_BO
- CM_RESOURCE_FAIL_NO_RETRY_BO
- CM_RESOURCE_FAILURE_RETRY_BO
- CM_CONV_DEALLOC_AFTER_SYNCPT
- CM_INCLUDE_PARTNER_REJECT_BO

4.42.3 State Changes

For half-duplex conversations, when *return_code* indicates **CM_OK:**

- The conversation enters receive state when Send_Data is issued with *send type* set to CM_SEND_AND_PREP_TO_RECEIVE and any of the following conditions are true:

— *Prepare_to_receive_type* is set to CM_PREP_TO_RECEIVE_FLUSH

— *Prepare_to_receive_type* is set to CM_PREP_TO_RECEIVE_CONFIRM

— *Prepare_to_receive_type* is set to CM_PREP_TO_RECEIVE_SYNC_LEVEL and *sync_level* is set to CM_NONE or CM_CONFIRM

— *Prepare_to_receive_type* is set to CM_PREP_TO_RECEIVE_SYNC_LEVEL, *sync_level* is set to CM_SYNC_POINT or CM_SYNC_POINT_NO_CONFIRM, but the conversation is not currently included in a transaction.

- The conversation enters **Defer-Receive** state when Send_Data is issued with *send type* set to CM_SEND_AND_PREP_TO_RECEIVE, *prepare_to_receive_type* set to CM_PREP_TO_RECEIVE_SYNC_LEVEL, *sync_level* set to CM_SYNC_POINT or CM_SYNC_POINT_NO_CONFIRM, and the conversation is included in a transaction.
- The conversation enters **Reset** state when Send_Data is issued with *send_type* set to CM_SEND_AND_DEALLOCATE and any of the following conditions is true:
  - *Deallocate_type* is set to CM_DEALLOCATE_ABEND
  — *Deallocate_type* is set to CM_DEALLOCATE_FLUSH
  - *Deallocate_type* is set to CM_DEALLOCATE_CONFIRM
  — Deallocate_type *is set to CM_DEALLOCATE_SYNC_LEVEL and* sync_level *is set to CM_NONE or CM_CONFIRM*
  — Deallocate_type *is set to CM_DEALLOCATE_SYNC_LEVEL, sync_level* is set to *CM_SYNC_POINT or CM_SYNC_POINT_NO_CONFIRM,* but the conversation is not currently included in a transaction.
- The conversation enters **Defer-Deallocate** state when Send_Data is issued with *send type* set to CM_SEND_AND_DEALLOCATE, *deallocate_type* set to CM_DEALLOCATE_SYNC_LEVEL. *sync_level* set to CM_SYNC_POINT or CM_SYNC_POINT_NO_CONFIRM, and the
- The conversation enters **Send** state when Send Data is issued in Send-Pending state with *send type* set to CM_BUFFER_DATA CM_SEND_AND_FLUSH or CM_SEND_AND_CONFIRM.
- No state change occurs when Send_Data is issued in Send state with *send_type* set to CM_BUFFER_DATA CM_SEND_AND_FLUSH, or CM_SEND_AND_CONFIRM.

For full-duplex conversations, when *return_code* indicates **CM_OK:**

- The conversation enters **Receive-Only** state when the Send_Data call is issued in **Send-Receive** state with *send type* set to CM_SEND_AND_DEALLOCATE**,** and one of the following conditions is true:
  — *deallocate_type* is set to CM_DEALLOCATE_FLUSH
  — *deallocate_type* is set to *CM_DEALLOCATE_SYNC_LEVEL* and *sync_level* is set to CM NONE.
  — *deallocate_type* is set to *CM_DEALLOCATE_SYNC_LEVEL, sync_level* is set to *CM_SYNC_POINT_NO_CONFIRM,* and the conversation is not currently included in a transaction.

The conversation enters **Reset** state when the **Send_Data** call is issued with *send type* set to CM_SEND_AND_DEALLOCATE and one of the following conditions is true:

— The call is issued in **Send-Only** state

— The call is issued in **Send-Receive**, **Send-Only**, **Sync-Point,** or **Sync-Point-Deallocate** state and *deallocate_type* is set to **CM_DEALLOCATE_ABEND**

- The conversation enters **Defer-Deallocate** state when the Send_Data call is issued with *send type* set to CM_SEND_AND_DEALLOCATE, *deallocate_type* set to CM_DEALLOCATE_SYNC_LEVEL**,** *sync_level* set to CM_SYNC_POINT_NO_CONFIRM**,** and the conversation included in a transaction.
- No state change occurs when Send_Data is issued in **Send-Receive** or **Send-Only** state with *send type* set to CM_BUFFER_DATA or CM SEND AND FLUSH.

4.42.4 Usage Notes

1. The local system may buffer data to be sent to the remote system until it accumulates a sufficient amount of data for transmission (from one or more Send_Mapped_Data calls), or until the local program issues a call that causes the system to flush its send buffer. The map routine is responsible for any buffering and determines when data should be sent. Therefore a flush call may or may not have an effect.

2. For a half-duplex conversation, when *control_information_received* indicates CM_REQ_TO_SEND_RECEIVED, CM_RTS_RCVD_AND_EXP_DATA_AVAIL, or CM_RTS_RCVD_AND_EXP_MAP_DATA_AVAIL or the remote program is requesting the local program's end of the conversation to enter Receive state, which places the remote program's end of the conversation in **Send** state. See 4.41, “Request_To_Send (CMRTS)” on page 212 for a discussion of how a program can place its end of a conversation in Receive state.

3 When issued during a mapped conversation, the Send_Mapped_Data call sends one complete data record. The data record consists entirely of data and CPI Communications does not examine the data for logical record length fields. It is this specification of a complete data record, at send time by the local program and what it sends, that is indicated to the remote program by the *data_received* parameter of the Receive_Mapped_Data call.

   For example, consider a mapped conversation where the local program issues two Send_Data calls with *send_length* set, respectively, to 30 and then 50. (These numbers are simplistic for explanatory purposes.) The local program then issues Flush and the 80 bytes of data are sent to the remote system. The remote program now issues Receive_Mapped_Data with *requested length* set to a sufficiently large value, say 1000. The remote program will receive back only 30 bytes of data (indicated by the *received_length* parameter) because this is a complete data record. The completeness of the data record is indicated by the *data_received* variable, which will be set to CM_COMPLETE_DATA_RECEIVED.

   The remote program receives the remaining 50 bytes of data (from the second Send_Mapped_Data) when it performs a second Receive_Mapped_Data with *requested_length* set to a value greater than or equal to 50.

   Note: The *received_length* may be different then the *send_length* due to differences in how data is locally represented in each partner system.

3. If map routine is unable to process the user data either because the user identified an incorrect *map_name* or the map routine encountered an error during

processing the *return_code* is set appropriately Nothing is sent nor is the conversation aborted, control is given back to the program for appropriate action.

4. The *send_length* is local information for the map routine. The send_length value that is used by the CRM is calculated by the map routine after all transformations are completed.
5. Send_Data is often used in combination with other calls, such as Flush, Confirm, and Prepare_To_Receive. Contrast this usage with the equivalent function available from the use of the Set_Send_Type call prior to issuing a call to Send_Data.
6. A program must not specify a value in the *send_length* parameter that is greater than the maximum the implementation can support. The maximum may vary from system to system. The maximum also depends on the *map_name* parameter.

   Every CPI-C product must be able to send buffers with a length of 32,767 bytes. Users who write map routines should describe the length of the data buffer before encoding, if the encoded buffer has a length of 32,767 bytes.

   The application program should not use a *send_length* larger than this,
7. When *control_information_received* indicates that expedited data is available to be received, subsequent calls with this parameter will continue to indicate that expedited data is available until the expedited data has been received by the program

4.42.5 Related Information

2.2. “Conversation Types” on page 11 provides more information on mapped and basic conversations.

3.3.1,“Data Buffering and Transmission” on page 65 provides a complete discussion of controls over data transmission.

All of the example program flows in Chapter 3, “Program-to-Program Communication Tutorial” make use of the Send_Data call.

For a mapped conversation with a non-null map name the largest amount of data a program can send in a Send_Mapped_Data call and the largest amount of data that can be received by the program in a Receive_Mapped_Data call depends on the map name.

4.38. "Receive (CMRCVM)” on page xxx provides more information on the *data_received* parameter.

4.71, "Set_Send_Type (CMSST)” on page 297 provides more information on the *send_type* conversation characteristic and the use of it in combination with calls to Send_Data.

SNA Transaction Programmers Reference Manual for LU Type 6.2 *provides further discussion of mapped conversations.*

Extract_Mapped_Initialization_Data (CMEMID)

A program using the OSI TP CRM can use the Extract_Mapped_Initialization_Data (CMEMlD) call to extract the *Initialization_data* and *initialization_data_length* conversation characteristics received from the remote program for a given conversation after they have been processed by the underlying map routine. The *map_name* identifies the data so that it can be properly processed. The values are returned to the program in the *Initialization_data* and *Initialization_data length* parameters.

The Extract_Mapped_Initialization_Data call is used by the recipient of the conversation to extract the incoming mapped initialization data received from the initiator of the conversation, and transformed by the underlying map routine. It may be issued following the Accept_Conversation or Accept_incoming call.

The Extract_Mapped_Initialization_Data call may also be used by the initiator of the conversation to extract the incoming initialization data from the recipient of the conversation after it has been processed by the U-ASE. In this case, the call may be issued following receipt of a *control_information_received* value of CM_ALLOCATE_CONFIRMED_WITH_DATA or CM_ALLOCATE_REJECTED_WITH_DATA.

Format

```
CALL CMEMID     (conversation_ID,
                 map_name,
                 map_name_length,
                 Initialization_data,
                 requested_length,
                 initialization_data_length,
                 return_code)
```

Parameters

conversation ID (input)
Specifies the conversation identifier.

map_name (output)

Specifies the variable containing the mapping function used to decode the data record. The length of the variable .must be at least 64 bytes.

map_name_length (output)

Specifies the variable containing the length of the returned map_name parameter.

Note: *Unless* return_code *is set to CM_OK, the value of* Initialization_data *is not meaningful.*

requested_length (input)
Specifies the length of the *Initialization_data* variable to contain the initialization data.

Initialization_data length (output)
If *return_code* is CM_OK, the output value of this parameter specifies the size of the *Initialization_data* variable in bytes. If *return_code* is CM_BUFFER_TOO_SMALL, this parameter indicates the size of the *Initialization_data* to be extracted.

return_code (output)
Specifies the result of the call execution. The *return_code* variable can have one of the following values:

- CM OK
- CM_CALL_NOT_SUPPORTED
- CM_PROGRAM_PARAMETER_CHECK

This value indicates one of the following:

— The *conversation_ID* specifies an unassigned identifier.

— The *requested_length* specifies a value less than 0.

— The conversation_type characteristic is set to CM_BASIC_CONVERSATION.

- CM_BUFFER_TOO_SMALL
  The *requested_length* specifies a value that is less than the size of the *Initialization_data* characteristic to be extracted. The *Initialization_data* characteristic is not returned, but the *Initialization_data length* parameter is set to indicate the size required.
- CM_UNKNOWN_MAP_NAME_RECEIVED
  The received data requires a map routine the local map function does not support. The buffer contains the unprocessed user data.
- CM_MAP_ROUTINE_ERROR
  The map routine encountered a problem with the received data. The buffer contains the unprocessed user data.
- CM_PROGRAM_STATE_CHECK

This value indicates that the *conversation_ID* specifies a conversation in **Initialize-incoming** state.

- CM OPERATION NOT ACCEPTED

This value indicates that the program has chosen conversation-level non-blocking for the conversation and a previous call operation is still in progress.

- CM_PRODUCT_SPECIFIC_ERROR

4.21.3 State Changes

This call does not cause any state changes.

4.21.4 Usage Notes

1. This call does not change the value of the *Initialization_data* or the *Initialization_data length* characteristic for the specified conversation.
2. The program that initiates the conversation (issues Initialize_Conversation) must set *allocate_confirm* to CM_ALLOCATE_CONFIRM if it is expecting initialization data to be returned from the remote program following its confirmation of acceptance of the conversation.
3. The call is not associated with any conversation queue. When a conversation uses queue-level non-blocking, the call does not return CM_OPERATION_NOT_ACCEPTED on the conversation.
4. The *received_length* is local information from the map routine. The received_length value is calculated by the map routine after all transformations are completed.
5. If CM_BUFFER_TOO_SMALL is returned *Initialization_data* will have been set to the length required to receive the data. The user can allocate a larger buffer in which to receive the data and reissue the call, or perform other user specified actions.

4.21.5 Related Information

4.58, "Set_Mapped_Initialization_Data (CMSMID)" on page xxx describes how the mapped initialization data is set by the remote program.

Set_Mapped_Initialization_Data (CMSMID)

Set_lnitialization_Data (CMSMlD) is used by a program using an OSI TO CRM to set the *Initialization_data* and *Initialization_data—length* conversation characteristics to be sent to the remote program for a given conversation. The *map_name* identifies the data so that the underlying map routine can properly transform it for transmission tot he remote partner. Set_Mapped_Initialization_Data overrides the values that were assigned when the lnitialize_Conversation, Accept_Conversation, Initialize_For_Incoming, or Accept_Incoming call was issued.

Format

```
CALL CMSMID     (conversation_ID,
                 map_name,
                 map_name_length,
                 Initialization_data,
                 initialization_data_length,
                 return_code)
```

Parameters

conversation ID (input)
Specifies the conversation identifier.

map_name (input)
Specifies the mapping function to be used to encode the data record. This is a 0 to 64 byte identifier.

map_name_length (input)
Specifies the length of the map_name. The length can be from 0 to 64 bytes. If zero, the *map_name* parameter is ignored.

lnitialization_data (input)
Specifies the initialization data that is to be passed to the remote program during conversation startup.

initialization_data_length (input)
Specifies the length of the initialization data. If zero, the *Initialization_data* parameter is ignored.

return_code (output)
Specifies the result of the call execution. The *return_code* variable can have one of the following values:

- CM OK
- CM_UNKNOWN_MAP_NAME_REQUESTED
  The supplied map name is unknown to the map routine, no data was sent.
- CM_MAP_ROUTINE_ERROR
  The map routine encountered a problem with the user data, no data was sent.
- CM_CALL_NOT_SUPPORTED
- CM_PROGRAM_STATE_CHECK

This value indicates one of the following:

— The conversation is not in initialize state, Receive state (for a half-duplex conversation), or **Send-Receive** state (for a full-duplex conversation).

- CM_PROGRAM_PARAMETER_CHECK This value indicates one of the following:
  — The *conversation_ID* specifies an unassigned identifier.
  — The *initialization_data_length* specifies a value greater than 10000 or less than zero.
  — The conversation_type characteristic is set to CM_BASIC_CONVERSATION.
- CM_OPERATION_NOT_ACCEPTED
- CM_PRODUCT_SPECIFIC_ERROR

4.58.3 State Changes

This call does not cause any state changes.

4.58.4 Usage Notes

1. Initialization data is data that the program initiating a conversation (using the Initialize_Conversation call) can choose to send to the remote program. The initiator issues a Set_Initialization_Data call between the Initialize_Conversation and Allocate calls to specify the initialization data to be sent by the Allocate call. The *map_name* parameter is set to identify the data so that the underlying map routine can properly handle it. Following the successful completion of either an Accept_Conversation or Accept_Incoming call, the remote program issues an Extract_Mapped_Initialization_Data call to extract the initialization data.

2. The remote program may issue a Set_Initialization_Data call to identify data that is to be sent to the initiating program on its next call following its acceptance of the conversation. To avoid overwriting initialization data from the initiating program, the recipient must issue the Extract_Mapped_Initialization_Data call to extract the incoming initialization data before issuing the Set_Mapped_Initialization_Data call. The Set_Initialization_Data must be issued before any other call on the conversation, except Set* or Extract* calls. To extract the data from the remote program, the initiating program issues the Extract_Mapped_Initialization_Data call after receiving a *control_information_received* value of CM_ALLOCATE_CONFIRMED_WITH_DATA or CM_ALLOCATE_REJECTED_WITH_DATA.

3. If a *return_code* other than CM_OK is returned on the call, the *Initialization_data* and *initialization_data_length* conversation characteristics remain unchanged.
4. If map routine is unable to process the user data either because the user identified an incorrect *map_name* or the map routine encountered an error during processing the *return_code* is set appropriately Nothing is sent nor is the conversation aborted, control is given back to the program for appropriate action.
5. Because of the work the local map routine performs, the real amount of data sent by the CRM may be larger than 10000 bytes, even if the initialization_data_length is less than 10000. In this case the remote system may not be able to process the data. So the maximum value of initialization_data_length depends on the map routine and the abilities of the remote system.
6. When the conversation is allocated using the LU 6.2 CRM, initialization data is sent as PIP data,

4.58.5 Related Information

4.21, "Extract_Mapped_Initialization_Data (CMEMID)” on page xxx describes the Extract_Mapped_Initialization_Data call.