SMARTdata UTILITIES

SC26-7091-01

Data Description and Conversion

IBM

SMARTdata UTILITIES

SC26-7091-01

Data Description and Conversion

Note!

Before using this information and the product it supports, be sure to read the general information under "Notices" on page vii.

| **Second Edition, April 1997**

| This edition applies to all platforms supported by SMARTdata UTILITIES Version 1 Release 2 and Version 2 Release
| 1, and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using
| the correct edition for the level of the product.

Publications are *not* stocked at the address below. Requests for IBM publications should be made to your IBM representative or the IBM branch office serving your locality.

You can order by calling IBM Software Manufacturing Solutions at 1-800-879-2755.

A form for reader comments is provided at the back of this publication. If the form has been removed, address your comments to:

International Business Machines Corporation
RCF Processing Department
G26/050
5600 Cottle Road
SAN JOSE, CA 95193-0000
U.S.A.

IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Contents

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to :

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
U.S.A.

Licensees of this program who wish to have information about it for the purpose of enabling (1) the exchange of information between independently created programs and other programs (including this one) and (2) the mutual use of the information that has been exchanged, should contact:

IBM Corporation
Information Enabling Requests
Dept. M13
5600 Cottle Road
San Jose, CA 95193

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Trademarks and service marks

The following terms, denoted by an asterisk (*) in this publication, are trademarks of the IBM Corporation in the United States or other countries or both:

AIX
IBM
MVS/ESA
Operating System/2
Operating System/400
OS/2
OS/400
VM/ESA

The following terms are trademarks of other companies:

Windows	Microsoft Corp.
Windows NT	Microsoft Corp.
Windows 95	Microsoft Corp.

About this book

This manual was written to help application programmers use the APIs used by the Data Description and Conversion (DD&C) program. DD&C:

1. Converts field data from one type to another. For example, to convert a binary to a floating point representation.
2. Converts data structures from the form in which they are stored by a specific language, compiler, or operating environment to one that can be accessed directly by a program written for a different language, compiler, or operating environment.

To work with DD&C, you should be familiar with the following:

- The operating system for your platform
- C programming language
- Any other programming languages that you may use when converting data, such as COBOL or PL/1, including their respective compilers and operating systems
- IBM's A Data Language (ADL), which describes the encodings of data types and structures.
- Character Data Representation Architecture (CDRA).

Bibliography

You can order books by calling IBM Software Manufacturing Solutions at 1-800-879-2755.

Table 1. SMARTdata UTILITIES for Windows Publications

Publication Title	Order Number
SMARTdata UTILITIES for Windows Set	SBOF-6135
SMARTdata UTILITIES for Windows Distributed FileManager User's Guide	SC26-7134
SMARTdata UTILITIES Data Description and Conversion	SC26-7091
SMARTdata UTILITIES VSAM Application Programming Interface Reference	SC26-7133
SMARTdata UTILITIES A Data Language Reference for Data Description and Conversion	SC26-7092

Table 2. SMARTdata UTILITIES for OS/2 Publications

Publication Title	Order Number
SMARTdata UTILITIES for OS/2 Set	SBOF-6131
SMARTdata UTILITIES for OS/2: VSAM in a Distributed Environment	SC26-7063
SMARTdata UTILITIES SMARTsort for OS/2 and AIX	SC26-7099
SMARTdata UTILITIES Data Description and Conversion	SC26-7091
SMARTdata UTILITIES A Data Language Reference for Data Description and Conversion	SC26-7092

Table 3. SMARTdata UTILITIES for AIX Publications

Publication Title	Order Number
SMARTdata UTILITIES for AIX Set	SBOF-6132
SMARTdata UTILITIES for AIX: VSAM in a Distributed Environment	SC26-7064
SMARTdata UTILITIES SMARTsort for OS/2 and AIX	SC26-7099
SMARTdata UTILITIES Data Description and Conversion	SC26-7091
SMARTdata UTILITIES A Data Language Reference for Data Description and Conversion	SC26-7092

Table 4 (Page 1 of 2). Other Publications

Publication Title	Order Number
DDM Architecture: Specifications for ADL	SC21-8286
Character Data Representation Architecture, Level 2	SC09-1390
IBM Systems Journal: Volume 31, No. 3, 1992	G321-5483
IBM Dictionary of Computing	SC20-1699
Compilers—Principles, Techniques, and Tools: by the Addison—Wesley Publishing Company	
IEEE Standard for Binary Floating—Point Arithmetic: ANSI/IEEE STANDARD	754-1985
INTEL 387™ DX User	

Table 4 (Page 2 of 2). Other Publications

Publication Title	**Order Number**
IBM Distributed Data Management: General Information	GC21-9527
IBM Distributed Data Management: Reference Guide	SC21-9526
Using Distributed Data Management for the IBM Personal Computer	SC21-9643
AS/400 Communications: Distributed Data Management Guide	SC21-9600
CICS/Distributed Data Management: User's Guide	SC33-0695
IBM 4680 Store Systems: Distributed Data Management: User's Guide	SC30-4915
DFSMS/MVS Version 1 Release 2 Distributed FileManager/MVS Guide and Reference	SC26-4915

Summary of Amendments

This section summarizes changes made for this edition of the publication.

April 1997

This book has been revised to make it platform independent and to add Windows support.

For OS/2, this book replaces *SMARTdata UTILITIES for OS/2 Data Description and Conversion*, SC26-7091-00. Information about the OS/2 trace function has been moved from Chapter 8 to Appendix D.

For AIX, this book replaces *SMARTdata UTILITIES for AIX Data Description and Conversion*, SC26-7066-00.

The basic information has not changed for either OS/2 or AIX.

Chapter 1. Data Conversion with DD&C

This chapter introduces DD&C and discusses:

- Why data conversion is necessary
- The types of data conversion DD&C performs
- The components used to perform data conversion

Why Data Conversion is Necessary

In today's distributed processing environment, it is often necessary to share data between heterogeneous operating systems and programming languages. In doing so, however, a major problem is differences in:

- The hardware environments,[1]
- The operating systems that the application programs run under,
- The programming languages in which the application programs are written,
- The method of encoding data that is carried out by the programming languages and their compilers.

Application programs that do not have all these factors in common are unlikely to be able to share their data. For example, an application program that is written in C language wanted to access a character string or binary digit that is stored by an MVS COBOL program. The data would not be in the proper representation for use by that language or operating system.

To solve this problem, an application programmer can write data conversion routines specifically to convert data from one format to the other. However, this approach is impractical given the continuing trend toward open computing systems. There are too many combinations of hardware, operating system, programming language, and compiler to make individual solutions feasible.

DD&C offers a solution to the data conversion problem. DD&C allows application programs to transparently read and exchange data with other programs that run in the same or different operating environments.

You can write applications in various programming languages for DD&C. programs that are written in:

- C
- COBOL
- FORTRAN
- PL/1

[1] Not all machines store numbers in the same way. We could use 32-bit integers as an example. On some machines, the lowest memory address contains the low-order byte of the integer. These machines are called “little endian”. On other machines, the lowest memory address contains the high-order byte of the integer. These machines are called “big endian”.

- Other high-level languages (HLL's)

and running under:

- AIX[2]
- MVS/ESA
- OS/2 (IBM Operating System/2)
- OS/400 (IBM Operating System/400)
- VM/ESA
- Windows NT and Windows 95

can share data with each other, provided the methods that are used to store data are known. In fact, as long as you know the data structure of a programming language, and the structure can be described by ADL, you can:

- Assign the proper CCSID (coded character set identifier) code, or
- Use ADL coding

to enable DD&C translation for different environments and languages.

Types of Data Conversions

DD&C carries out planned conversions and ad hoc conversions for structured data and individual data fields.

Planned conversions of structured data
You can convert record-oriented, structured data by performing a planned conversion.

Planned conversions of data fields
You can also use the **planned** method of data conversion to:

- Convert data fields
- Change the ordering of data fields, or
- Produce a reduced view of data

when not all fields of the original data need to be represented in the converted data.

Ad hoc conversions of data fields
You can convert individual data fields from one type to another by performing an **ad hoc conversion**. You do this by calling the appropriate DD&C for Windows data-type conversion routines from within your application program.

Using Your Own Conversion Routines

DD&C also includes a **user-exit** function whereby you use a CALL statement to run your own conversion routines during program run time. The user-exit is described in Chapter 7.

[2] DD&C does not serialize the processing of DD&C functions within threads of the same process, or threads from different processes.

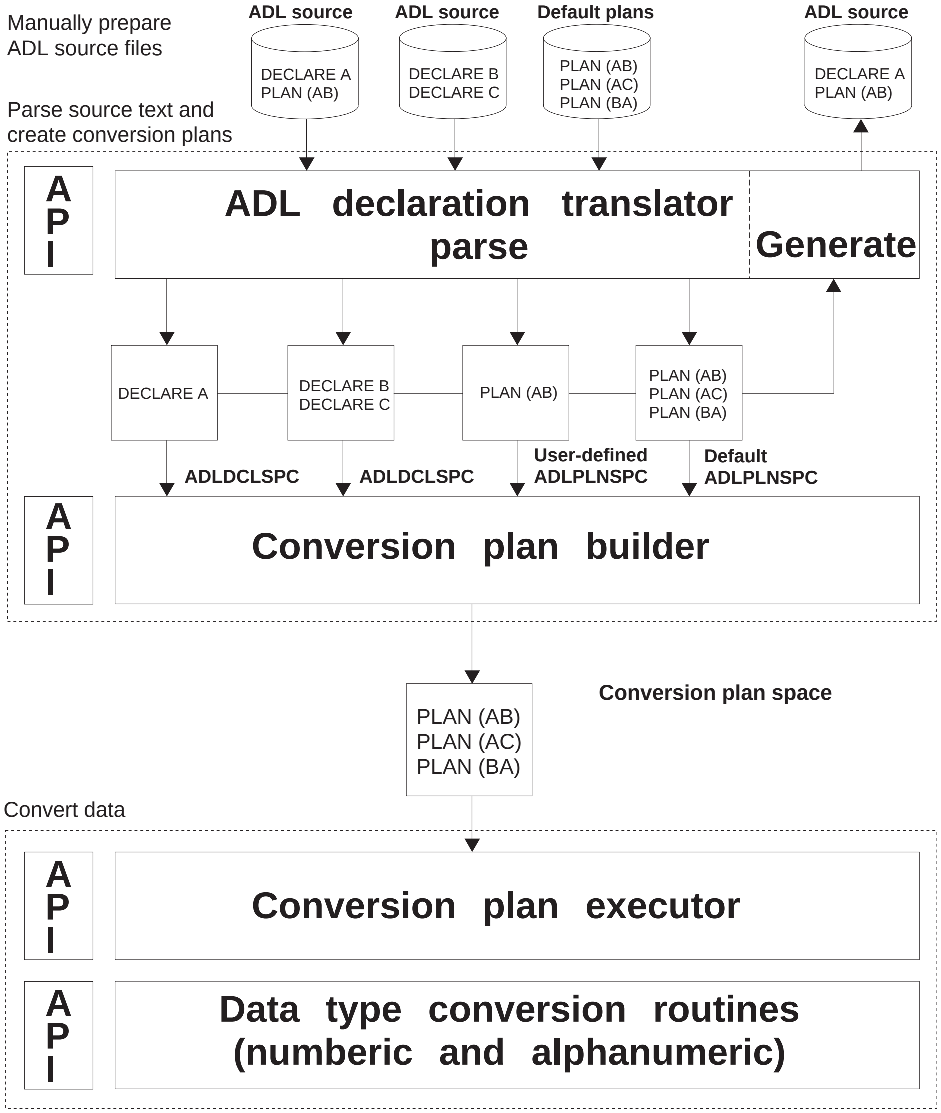


Figure 1. DD&C Components

Planned Conversions

A number of functionally distinct components perform data conversion. Each component has associated application programming interfaces (APIs) or functions. These APIs or functions are called by your application program. The components involved in data conversion are described on the following pages and in Figure 2 through Figure 3. The APIs that are used by each component are described in:

Chapter 3, “The ADL Declaration Translator APIs” on page 27
Chapter 4, “The Conversion Plan Builder API” on page 39
Chapter 5, “The Conversion Plan Executor APIs” on page 49
Chapter 6, “The Data-Type Conversion Routines” on page 59
Chapter 7, “The User Exit — Calling Your Own Programs” on page 107.

The steps involved in planned data conversions are:

1. Create the ADL source files to describe the physical layout of the source and target data by using ADL.
2. Start the parse function of the ADL declaration translator to convert the ADL data descriptions into encoded forms. The encoded statements are automatically stored in memory.
3. Start the conversion plan builder which uses the encoded forms of ADL data descriptions to build a conversion plan that maps the source data onto the target data.
4. Execute the conversion plan at run time to convert the data for one or more data structures.
5. Start the data-type conversion routines to convert numeric and alphanumeric data.

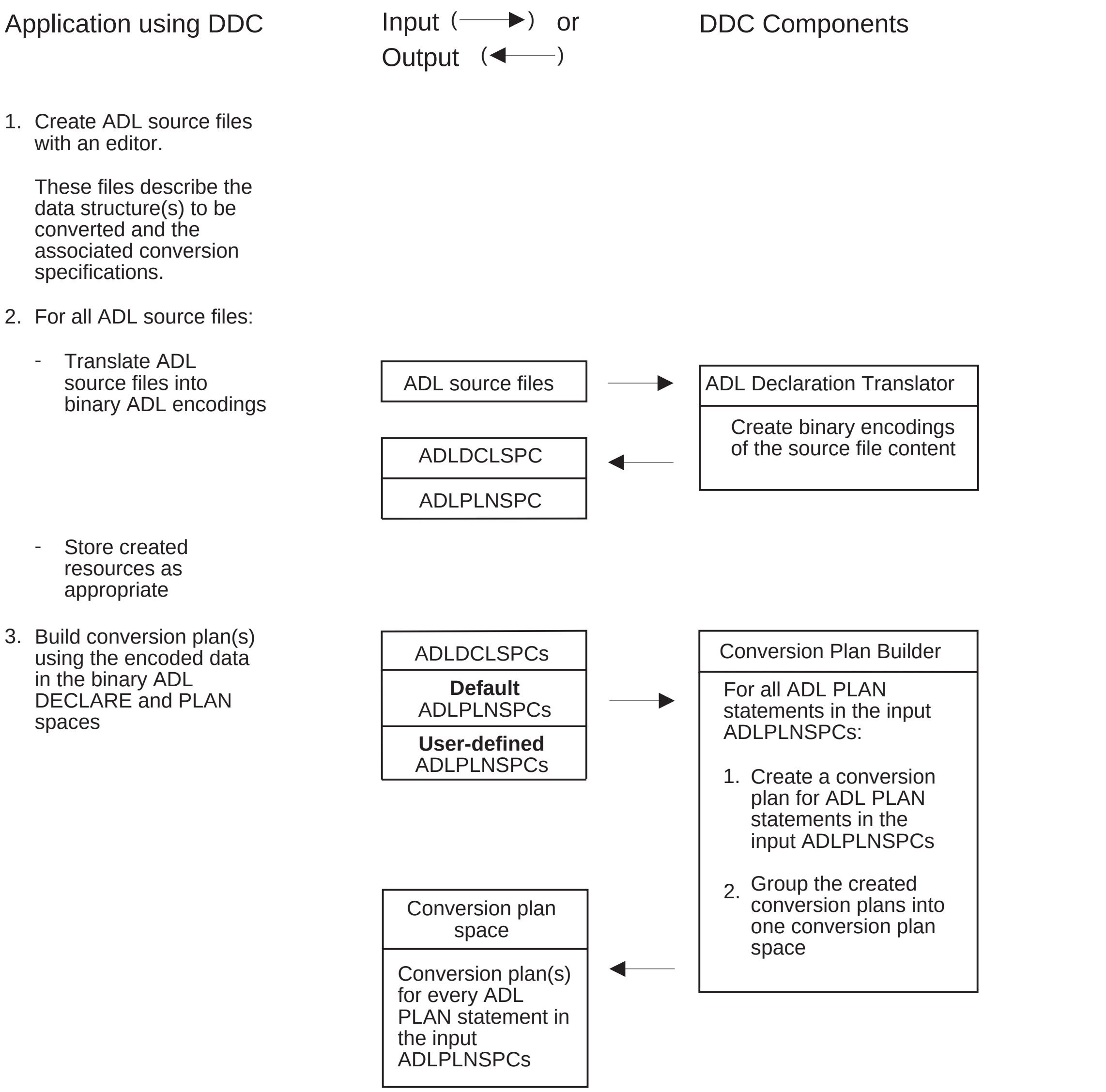

Application using DDC
Input (→) or
Output (←)
DDC Components
1. Create ADL source files with an editor.
These files describe the data structure(s) to be converted and the associated conversion specifications.
2. For all ADL source files:
- Translate ADL source files into binary ADL encodings
ADL source files
ADL Declaration Translator
Create binary encodings of the source file content
ADLDCLSPC
ADLPLNSPC
- Store created resources as appropriate
3. Build conversion plan(s) using the encoded data in the binary ADL DECLARE and PLAN spaces
ADLDCLSPCs
Default ADLPLNSPCs
User-defined ADLPLNSPCs
Conversion Plan Builder
For all ADL PLAN statements in the input ADLPLNSPCs:
1. Create a conversion plan for ADL PLAN statements in the input ADLPLNSPCs
2. Group the created conversion plans into one conversion plan space
Conversion plan space
Conversion plan(s) for every ADL PLAN statement in the input ADLPLNSPCs

Figure 2. Creating and Executing Planned Conversions, Part 1

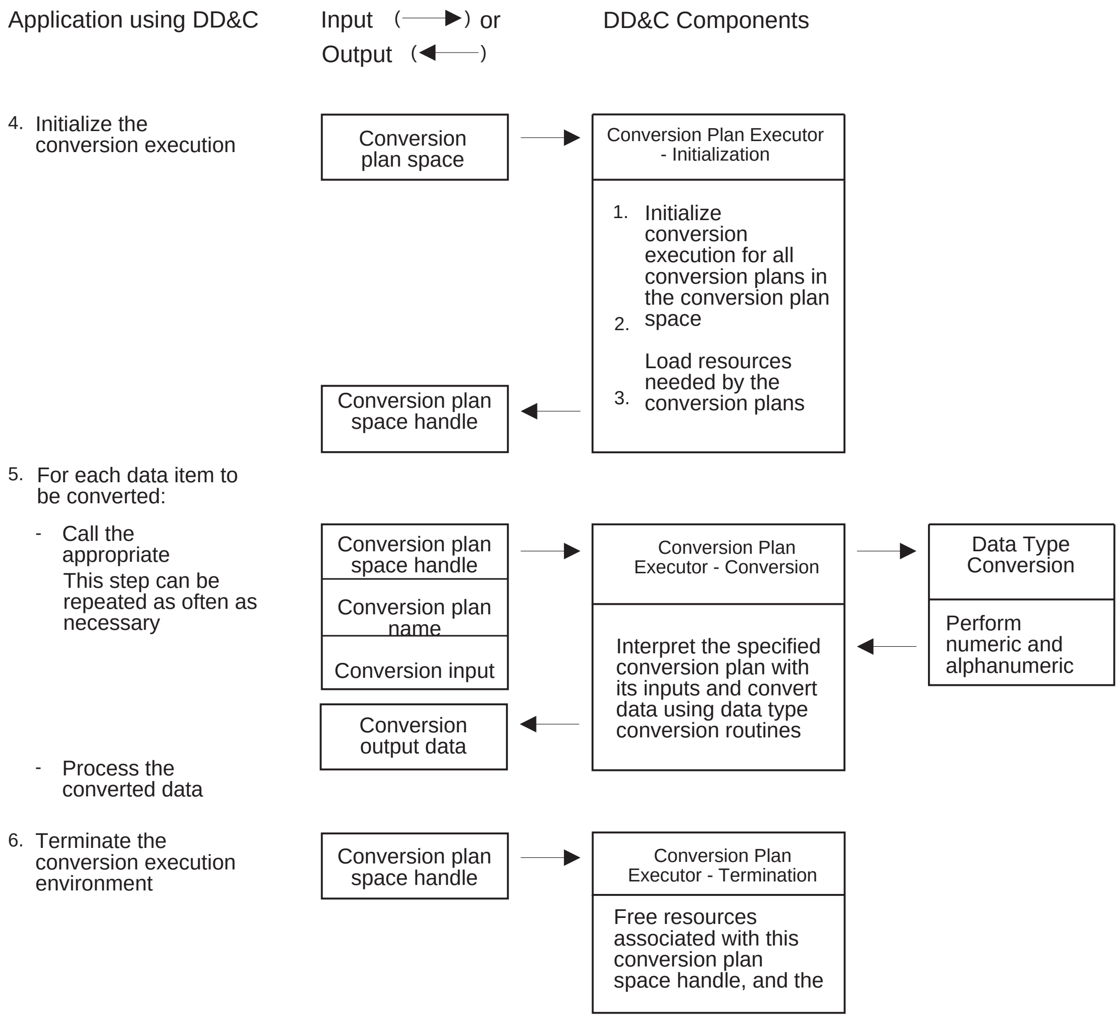

Application using DD&C
Input ( ) or
Output ( )
DD&C Components
4. Initialize the conversion execution
Conversion plan space
Conversion Plan Executor - Initialization
1. Initialize conversion execution for all conversion plans in the conversion plan space
2. Load resources needed by the conversion plans
3.
Conversion plan space handle
5. For each data item to be converted:
- Call the appropriate
This step can be repeated as often as necessary
Conversion plan space handle
Conversion plan name
Conversion input
Conversion Plan Executor - Conversion
Interpret the specified conversion plan with its inputs and convert data using data type conversion routines
Data Type Conversion
Perform numeric and alphanumeric
Conversion output data
- Process the converted data
6. Terminate the conversion execution environment
Conversion plan space handle
Conversion Plan Executor - Termination
Free resources associated with this conversion plan space handle, and the

Figure 3. Creating and Executing Planned Conversions, Part 2

Creating ADL Source Files

The first step in the conversion process is to create ADL source files. You do this by writing ADL source text by using a standard text editor. Refer to *SMARTdata UTILITIES A Data Language Reference for Data Description and Conversion* for the syntax and semantics of the ADL data description language.

The ADL source files must accurately describe the physical memory layout of both the source data and target data. The source data is the data that is to be converted and the target data is the format that the data must have after conversion. The ADL source files must also specify how to map the source data onto the target data.

The parse function of the ADL Declaration Translator receives the ADL source files as input.

The ADL Declaration Translator

The figure below shows how the ADL Declaration Translator operates.

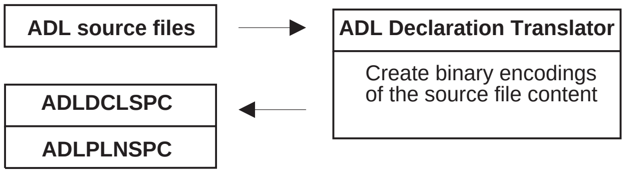

ADL source files
ADL Declaration Translator
Create binary encodings of the source file content
ADLDCLSPC
ADLPLNSPC

Figure 4. The ADL Declaration Translator

The ADL Declaration Translator component consists of two APIs, or functions:

1. The **Parse** function, FMTPRS, creates encoded ADL statements. FMTPRS uses files that describe the physical layout of the source and target data. A benefit from FMTPRS is that by doing this compilation step now, you will get better run-time performance than you would if the ADL source were interpreted at run time.

   The encoded ADL statements are stored in blocks of memory that is called:

   ADLDCLSPC This is the ADL declare space that describes the data. Normally there are two ADL declare statements; one describing the source data and one the target data. This results in two ADLDCLSPC's.

   ADLPLNSPC This is the ADL plan space that specifies how to map the source data onto the target data.

2. The **Generate** function, FMTGEN, allows you to use an existing ADL file as a template to change an ADL file. You can also create your own ADL file from an existing ADLDCLSPC OR ADLPLNSPC object.

   It is also a helpful tool for troubleshooting. **Generate** will show you the ADL source statements from which the ADLDCLSPC or ADLPLNSPC was created. You can spot any incorrect statements, change them, and then rerun the **parse** function to create a new ADLDCLSPC or ADLPLNSPC.

The Conversion Plan Builder

The figure below shows how the Conversion Plan Builder operates.

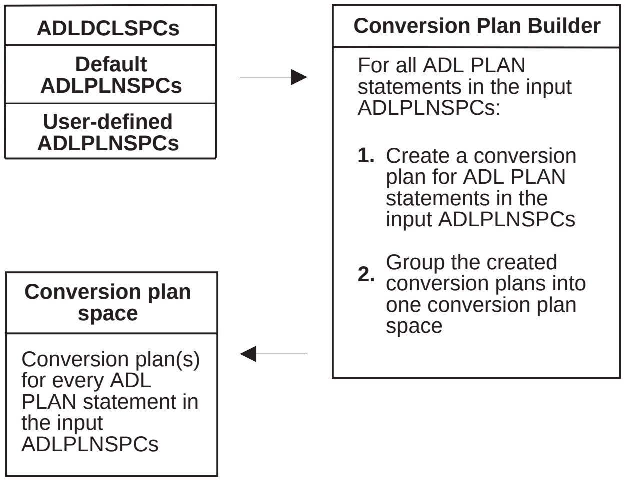

ADLDCLSPCs
Default ADLPLNSPCs
User-defined ADLPLNSPCs
Conversion Plan Builder
For all ADL PLAN statements in the input ADLPLNSPCs:
1. Create a conversion plan for ADL PLAN statements in the input ADLPLNSPCs
2. Group the created conversion plans into one conversion plan space
Conversion plan space
Conversion plan(s) for every ADL PLAN statement in the input ADLPLNSPCs

Figure 5. The Conversion Plan Builder

The Conversion Plan Builder accepts as input the blocks of memory that contain the encoded ADLPLNSPC and ADLDCLSPC spaces. The ADLPLNSPC and ADLDCLSPC spaces are generated by the parse function of the ADL Declaration Translator. It generates a conversion plan for each plan statement. The ADL plan space can contain one or more plan statements. Likewise, the conversion plan space that was created by the conversion plan builder, can contain one or more conversion plans. Figure 2 on page 5 shows the process flow for creating a conversion plan. ADLPLNSPCs are either:

- **Default**. At least one default ADLPLNSPC must be specified.
- **User-defined**. User-defined ADLPLNSPCs are optional.

If you do not specify user-defined plans, the Conversion Plan Builder uses the default plan or plans to generate the conversion plan for an application.

If you specify a user-defined plan that has the same name as a default plan, the Conversion Plan Builder uses it instead of the default plan.

If you specify a user-defined plan but no default plan with the same name exists, the Conversion Plan Builder returns a warning and no conversion plan is created.

The Conversion Plan Builder stores all conversion plans in a single *conversion plan space*.

The Conversion Plan Executor

The figure below shows how the Conversion Plan Executor operates.

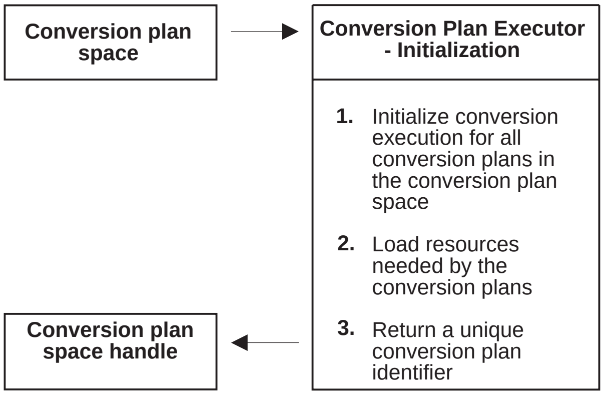

Conversion plan space
Conversion Plan Executor - Initialization
1. Initialize conversion execution for all conversion plans in the conversion plan space
2. Load resources needed by the conversion plans
3. Return a unique conversion plan identifier
Conversion plan space handle

Figure 6. The Conversion Plan Executor – Initialization

The Conversion Plan Executor component accomplishes the actual conversion of the source data into the target data according to the specifications in the conversion plan. Start the Conversion Plan Builder APIs at run time to:

- **Initialize** the Conversion Plan Executor. FMTCPXI loads the required conversion plan space into memory and initializes the resources for all conversion plans in the conversion plan space, including conversion tables.
- **Perform** the conversion. FMTCPXC converts data according to the specifications in the conversion plan. You can call it multiple times.
- **Terminate** the Conversion Plan Executor. FMTCPXT releases the resources that were loaded during initialization.

The Conversion Plan Executor converts data that is based on the conversion plans that is created by the Conversion Plan Builder. To achieve maximum data throughput at conversion time, use separate initialization and termination functions as shown in Figure 3 on page 6.

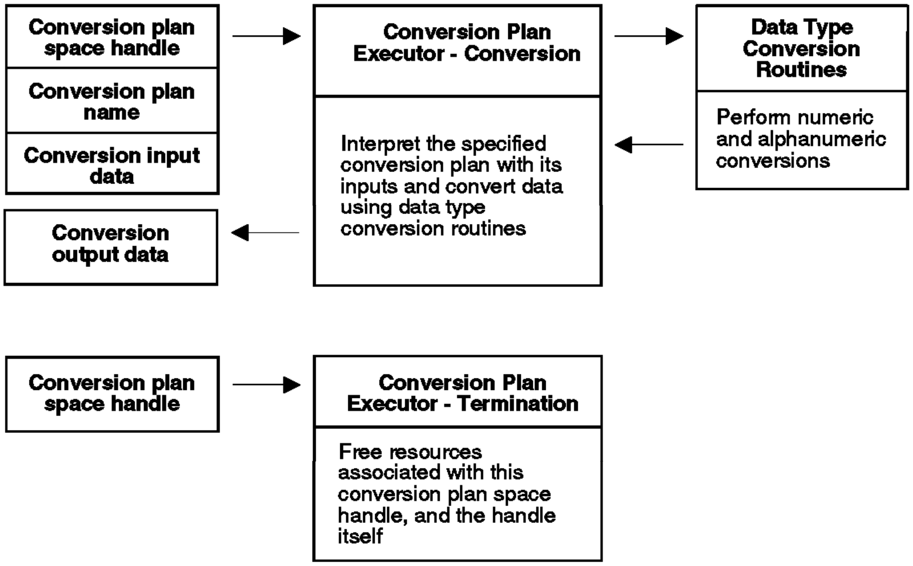
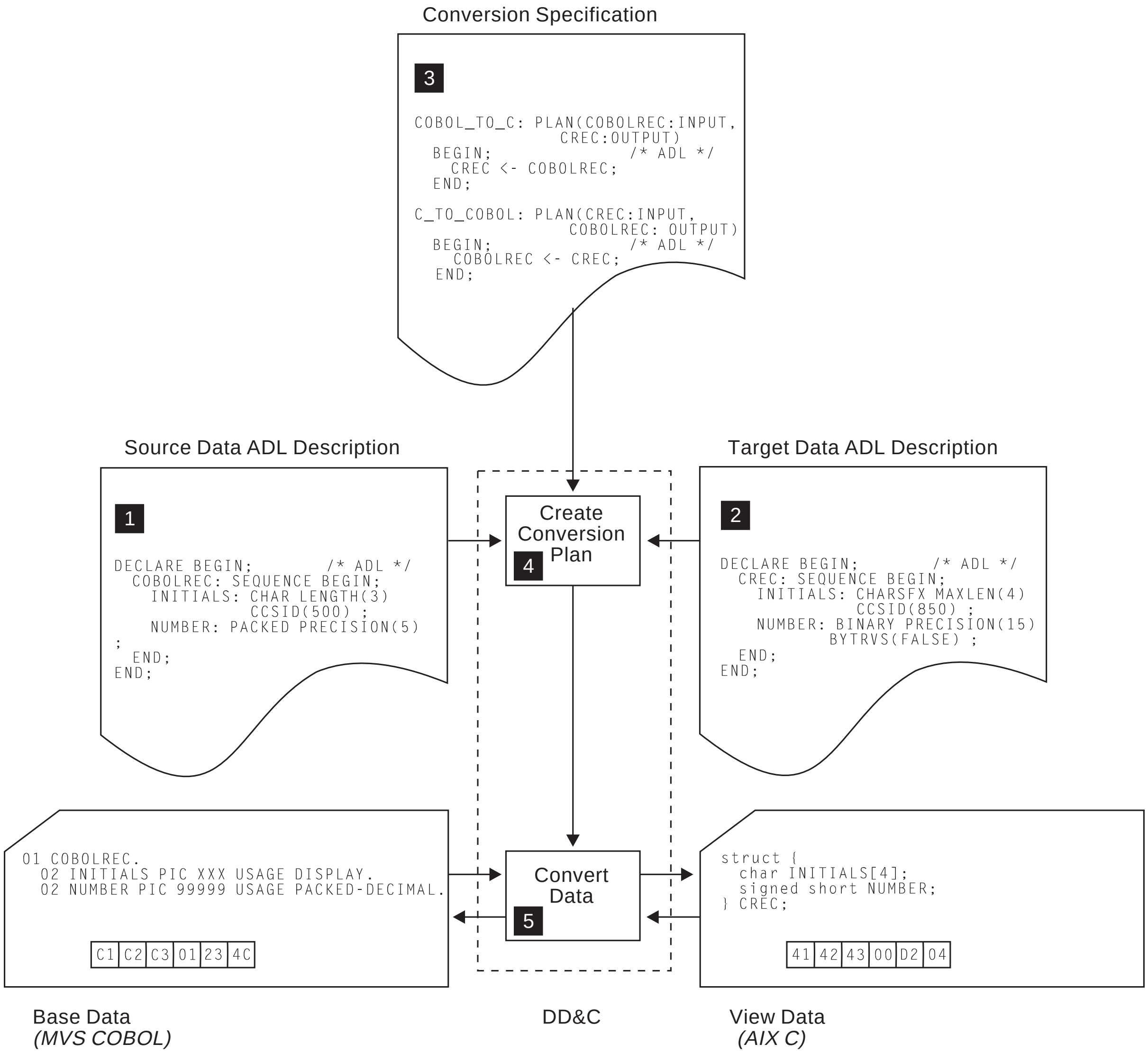

Conversion plan space handle
Conversion plan name
Conversion input data
Conversion output data
Conversion Plan Executor - Conversion
Interpret the specified conversion plan with its inputs and convert data using data type conversion routines
Data Type Conversion Routines
Perform numeric and alphanumeric conversions
Conversion plan space handle
Conversion Plan Executor - Termination
Free resources associated with this conversion plan space handle, and the handle itself

Figure 7. The Conversion Plan Executor - Conversion and Termination

The Data-Type Conversion Routines: Data-type conversion routines can be used by both planned conversion called at any time during planned or ad hoc data routines. You use data-type conversion routines to convert alphanumeric and numeric field data. The data-type conversion routines are described in 14.

Example of Planned Conversion

Figure 8 shows an example of a planned conversion. The numbers indicate the sequence of actions.



```
COBOL_TO_C: PLAN(COBOLREC:INPUT,
                 CREC:OUTPUT)
  BEGIN;                /* ADL */
    CREC <- COBOLREC;
  END;

C_TO_COBOL: PLAN(CREC:INPUT,
                  COBOLREC: OUTPUT)
  BEGIN;                /* ADL */
    COBOLREC <- CREC;
   END;
```

```
DECLARE BEGIN;        /* ADL */
  COBOLREC: SEQUENCE BEGIN;
    INITIALS: CHAR LENGTH(3)
               CCSID(500) ;
    NUMBER: PACKED PRECISION(5)
;
  END;
END;
```

```
DECLARE BEGIN;        /* ADL */
  CREC: SEQUENCE BEGIN;
    INITIALS: CHARSFX MAXLEN(4)
               CCSID(850) ;
    NUMBER: BINARY PRECISION(15)
            BYTRVS(FALSE) ;
  END;
END;
```

```
01 COBOLREC.
  02 INITIALS PIC XXX USAGE DISPLAY.
  02 NUMBER PIC 99999 USAGE PACKED-DECIMAL.
```

```
struct {
  char INITIALS[4];
  signed short NUMBER;
} CREC;
```

Figure 8. Example of Planned Conversion

Suppose that an AIX application that is written in C needs to read data that is generated by an MVS application that is written in COBOL. The data to be shared is:

```
"ABC", +1234
```

An MVS COBOL program describes the data as:

```
01 COBOLREC.
   02 INITIALS PIC XXX USAGE DISPLAY.
   02 NUMBER PIC 99999 USAGE PACKED-DECIMAL.
```

The following steps enable an AIX C program to access this data:

Step 1 Describe the physical memory layout of the source data in the MVS COBOL program using ADL DECLARE statements. This description subsequently serves as input for the conversion process.

```
/* Description of the source data */

DECLARE BEGIN;         /* ADL */
  COBOLREC: SEQUENCE BEGIN;
    INITIALS: CHAR LENGTH(3) CCSID(500);
    NUMBER: PACKED PRECISION(5);
  END;
END;
```

The source file record in this example consists of the following two fields:

INITIALS A string of 3 characters with CCSID 500.

NUMBER A decimal numeric field with 5 significant decimal digits.

Step 2 Describe the physical representation of the target data that is expected by an AIX C program. This is done using ADL DECLARE statements. The description completely identifies the physical memory layout of the target data that is returned by the conversion process. This format represents the data in a way that the Windows C program can process it directly.

```
/* Description of the target data */

DECLARE BEGIN;          /* ADL */
   CREC: SEQUENCE BEGIN;
     INITIALS: CHARSFX MAXLEN (4) CCSID(850);
     NUMBER: BINARY PRECISION(15) BYTRVS (FALSE);
  END;
END;
```

The target file consists of the same fields as the source file with different attributes:

INITIALS A string of characters with CCSID 850.
NUMBER A decimal numeric field with 15 significant decimal digits.

Note: This example is the same for OS/2 and Windows except for the BYTRVS value which is TRUE.

Step 3 Describe the method of converting the source data to the target data (the *conversion specification*):

```
/* Conversion specification from MVS COBOL to AIX C */
COBOL_TO_C: PLAN (COBOLREC: INPUT
                  CREC: OUTPUT)
         BEGIN;         /* ADL */
             CREC <- COBOLREC;
         END;
/* Conversion specification from to AIX C to MVS COBOL */

C_TO_COBOL: PLAN (CREC: INPUT,
              COBOLREC: OUTPUT)
         BEGIN;         /* ADL */
            COBOLREC <- CREC;
         END;
```

ADL conversion specifications consist of assignment statements between ADL PLAN statements.

Note: Conversion specifications permit both unidirectional and bidirectional conversions. This example describes a bidirectional conversion, where the MVS COBOL data is once the source (`COBOL_TO_C`) and once the target (`C_TO_COBOL`).

Step 4 Create a conversion plan, supplying the source-data description, the target-data description, and the conversion specifications as inputs. See "The Conversion Plan Builder" on page 7 for further details of how this is accomplished.

Step 5 Perform the data conversion, using the conversion plan to convert the source data and generate the target data. The Windows C program can read the target data and has the following format:

```
struct {
  char INITIALS[4];
  signed short NUMBER;
  } CREC;
```

Ad Hoc Conversions

The second method of converting data is by calling the DD&C data-type conversion routines directly from within an application program.

DD&C does not provide conversion routines for all possible combinations of data types for two reasons:

1. DD&C does not support conversion between a particular pair of data types, such as CHAR to BINARY.

2. A particular numeric conversion is done by some other means, such as ASIS to ASIS (simple copy), and ENUMERATION to ENUMERATION. Other numeric routines cover these conversions.

SMARTdata UTILITIES A Data Language Reference for Data Description and Conversion describes which data-type conversions are possible with DD&C.

Data-Type Conversion Routines — Numeric

Table 5 shows the conversion routines that are supplied by DD&C for numeric data types.

Table 5. Numeric Conversion Routines

To → **↓ From**	**BINARY**	**FLOAT**	**PACKED**	**ZONED**
BINARY	FMTBNBN	FMTBNFL	FMTBNPK	FMTBNZN
FLOAT	FMTFLBN	FMTFLFL	FMTFLPK	FMTFLZN
PACKED	FMTPKBN	FMTPKFL	FMTPKPK	FMTPKZN
ZONED	FMTZNBN	FMTZNFL	FMTZNPK	FMTZNZN

The field data types in Table 5 are as follows:

BINARY A fixed-length string of bits that encodes a fixed-precision number in dual representation.

FLOAT A fixed-length string of bits that encodes a number by means of a characteristic and a significand.

PACKED A fixed-length string of bits. Each byte represents two decimal digits of the represented number.

ZONED A fixed-length string of bits. Each byte represents one decimal digit of the represented number.

Data-Type Conversion Routines — Alphanumeric

To perform alphanumeric conversions, DD&C uses a subset of the services that is provided by the Character Data Representation Architecture (CDRA). CDRA services that are used by DD&C include:

- CDRA identifiers
- CDRA resources
- Common services.

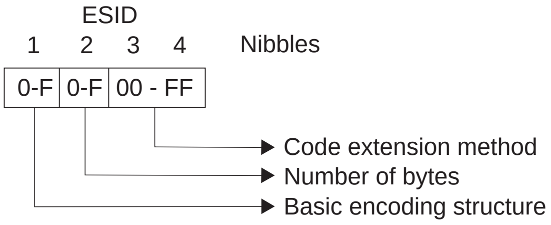
CDRA Identifiers Used in DD&C

CDRA identifiers provide the means by which a graphic character associated with a code point can be determined unambiguously.[3]

The following CDRA identifiers are used in DD&C API calls. These descriptions do not include the values that are allowed for each field. For that information, see *Character Data Representation Architecture, Level 2*.

- The ESID - encoding scheme identifier.

  The ESID is a 4-digit hexadecimal number that uniquely identifies a particular encoding scheme.



  The ESID has the following elements:

  1. The first nibble [4] contains the basic encoding structure. This element identifies the basic structural characteristic that differentiates various encoding schemes, such as EBCDIC (extended binary-coded decimal interchange code), ISO-8, IBM-PC data, and others.
  2. The second nibble contains the number of bytes in each code point. An encoding scheme might allow variations in the number of bytes that are associated with a code point. For example, SBCS (single-byte character set) or DBCS (double-byte character set) allow variations in the number of bytes.
  3. The last two nibbles contain the *code extension* method. Code extensions are techniques that are used to encode more characters than can be accommodated in the basic encoding structure. An example is the use of SO (shift-out) and SI (shift-in) as control characters. These control characters are used to access an alternative assignment of graphic characters to code points. The SO and SI control characters show whether one or two bytes of data constitute a code point in the EBCDIC mixed SBCS and DBCS code. This element of the ESID identifies the particular method of code extension used from among the many that are possible in the encoding scheme.

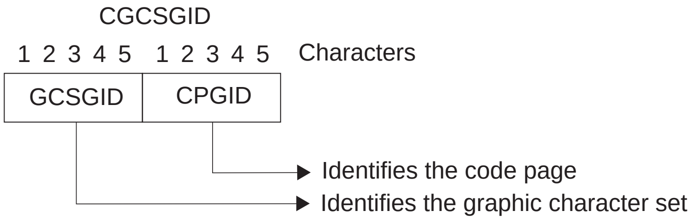
  For details of the values that are allowed, refer to the *Character Data Representation Architecture, Level 2*.

- The CGCSGID - coded graphic character set global identifier.

[3] Each unique bit pattern that is defined by a code is called a code point.

[4] A nibble is a sequence of four bits, that is, half a byte.

The CGCSGID has two decimal numbers:


CGCSGID
1 2 3 4 5 1 2 3 4 5 Characters
GCSGID
CPGID
Identifies the code page
Identifies the graphic character set

1. The graphic character set global identifier (GCSGID) A 5-digit decimal number that uniquely identifies a graphic character set. A character set is a defined set of characters. No coded representation is assumed.
2. The code page global identifier (CPGID). A 5-digit decimal number assigned to a code page. A code page is a specification of code points for each graphic character in a character set, or in a collection of graphic character sets. Within a given code page, a code point can have only one meaning. The same code pages are used when the encoding structures are similar but not identical.

- The CCSID - coded character set identifier.

  This 16-bit number identifies a unique combination of:

  - Encoding scheme identifier (ESID)
  - Character set identifier or identifiers
  - Code page identifier or identifiers
  - Additional coding-related information that uniquely identifies the coded graphic character representation that is used.

CDRA Services Used by DD&C

DD&C implements a subset of the data conversion services that is defined by CDRA, including the following alphanumeric conversion routines.

- APIs that retrieve information from a CCSID resource table are:

  CDRGESE Get an encoding scheme element and its subelements

  CDRSMXC Get the CCSID with largest character set for a specified encoding scheme and code page

  CDRGESP Get encoding scheme, character set, and code page elements

  CDRGCTL Get the definition of control function.

- APIs that convert characters from one CCSID to another are:

  CDRMSCI Initialize multiple-step conversion
  CDRMSCP Perform multiple-step conversion
  CDRMSCC Clean up after multiple-step conversion.

Chapter 6, “The Data-Type Conversion Routines” describes these alphanumeric conversion APIs in detail.

CDRA Resources Used by DD&C

CDRA resources are machine representations of tables that are:

- Associated with graphic character data conversion
- Used for querying defaults
- Used for finding relationships between different CCSIDs.

The CDRA resources that are used by DD&C are:

CCSID resource table
The CCSID resource table contains representations of the various elements that are associated with a CCSID within a system in a machine-readable form.

Graphic character conversion selection table
This table relates the source and target CCSID to the correct conversion table and conversion algorithm.

Graphic character conversion tables
These tables contain “CCSID to CCSID” conversion strings. The conversion strings consist of hexadecimal code points for the special conversion.

For example, the CCSID resource table identifies the related CPGID. You can append the CCSID to an ADL data description of a field or a file. If both the source and target CPGID are used for the conversion of a character field, both CPGIDs are used to look up the applicable conversion table.

You must select and use the correct conversion tables to ensure that data objects can be shared between different computing environments.

Select the conversion method according to the encoding scheme and string type of the input and output data. The string types are:

- Input that ends with a null character
- Output that ends with a null character
- Output space padded

Substring separation might be necessary if the input data string contains embedded code extension controls. Character controls such as shift-in (SI) or shift-out (SO) in EBCDIC mixed single-byte character set (SBCS) and double-byte character set (DBCS) data might require substring separation. The conversion method first parses the input data string and, if necessary, performs any required *substring separation*.

DD&C supports SBCS to SBCS, DBCS to DBCS, and SBCS to DBCS conversions.

The required CDRA resources are listed for each of the alphanumeric data conversion routines that are described in Chapter 6.

Note: You cannot add to, change, or delete any part of a CDRA resource from within DD&C.

Chapter 2. DD&C Data Areas and Data Structures

The DD&C data areas and data structures described in this chapter are:

- Conventions
- The ADL Communication Area – FMTADLCA
- The Condition Token – FMTCTOK
- The Consistency Token – FMTCNSTKN

Conventions

Some conventions to observe when calling API functions in DD&C are the data definitions used by the APIs and Hungarian Notation to specify variable data.

Data types

Table 6 describes data types used in the DD&C APIs, their definitions and platform equivalents.

Table 6. Data Types and Definitions

Type	Definition	Platform Equivalent
LONG	Signed integer in the range -2 147 483 648 through 2 147 483 647	typedef long LONG;
SHORT	Signed integer in the range -32 768 through 32 767	typedef short SHORT;
UCHAR	Unsigned integer in the range 0 through 255	typedef unsigned char UCHAR;
UINT	Unsigned integer in the range 0 through 4 294 967 295	typedef unsigned int UINT;
ULONG	Unsigned integer in the range 0 through 4 294 967 295	typedef long ULONG;
USHORT	Unsigned integer in the range 0 through 65 535	typedef unsigned short USHORT;
PBYTE	Pointer to a data area	typedef unisgned char *PBYTE;
PCHAR	Pointer to a character	typedef char *PCHAR;
PLONG	Pointer to a signed integer	typedef long *PLONG;
PSHORT	Pointer to a signed short integer	typedef short *PSHORT;
PULONG	Pointer to an unsigned integer	typedef unsigned long *PLONG;
PUSHORT	Pointer to an unsigned short integer	typedef unsigned short *PSHORT;
PVOID	Pointer to a data type of undefined format.	typedef void *PVOID;

Hungarian Notation

Hungarian notation is used for DD&C variable names. This naming convention adds a lowercase prefix abbreviation of the data type to the beginning of the variable so that its type is immediately recognizable. The prefixes used in the APIs are:

ab	Array of bytes
ap	Array of pointers
l	Long (LONG)
p	Pointer
pch	Pointer to a single-byte character string.
pp	Pointer to a pointer
s	Short (SHORT)
uch	Unsigned character (UCHAR)
ui	Unsigned integer (UINT)
ul	Unsigned long (ULONG)
us	Unsigned short (USHORT)

The Condition Token

All DD&C API functions return a condition token to indicate their current processing status. The name of the field is FMTCTOK and it's pointed to by the *pfeedback* parameter. To declare and initialize a condition token, use the following format.

```
FMTISINF MyIsInfo;
FMTCTOK  MyCtok={{0,0},0,0,0,"",&MyIsInfo};
```

The condition token is 12 bytes long, it's format is shown in Figure 9.

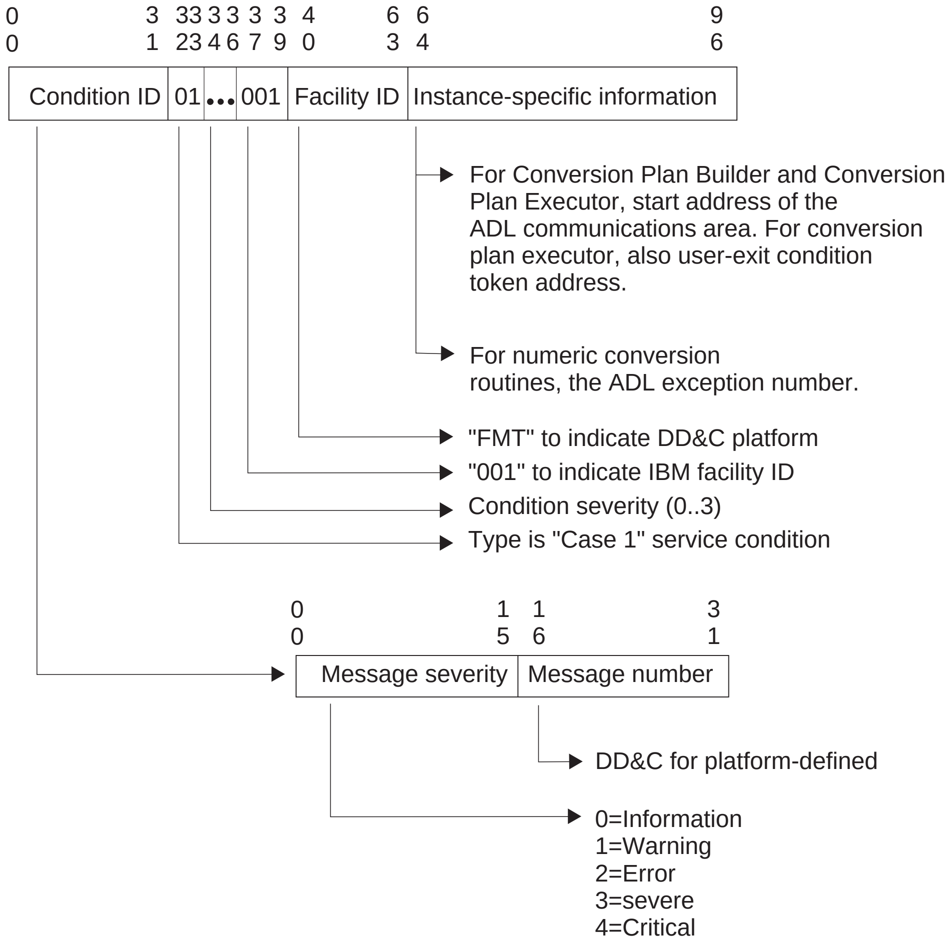

0 0
3 1
3 2
3 3
3 4
3 6
3 7
3 9
4 0
6 3
6 4
9 6
Condition ID
01
•••
001
Facility ID
Instance-specific information
For Conversion Plan Builder and Conversion Plan Executor, start address of the ADL communications area. For conversion plan executor, also user-exit condition token address.
For numeric conversion routines, the ADL exception number.
"FMT" to indicate DD&C platform
"001" to indicate IBM facility ID
Condition severity (0..3)
Type is "Case 1" service condition
0 0
1 5
1 6
3 1
Message severity
Message number
DD&C for platform-defined
0=Information
1=Warning
2=Error
3=severe
4=Critical

Figure 9. Format of the DD&C Condition Token

```
typedef struct _FMTCTOK
{
   struct
   {
      USHORT usMsgSev;
      USHORT usMsgNo;
   }  Condition_ID;
   UINT   fCase    :2;
   UINT   fSeverity:3;
   UINT   fControl :3;
   UCHAR  uchFacility_ID[3];
   union
   {
     ULONG            ulAdlExId;
     FMTADLCA         *pAdlCommArea;
     struct _FMTCTOK   *pUserExitCtok;
   } pI_S_Info;
} FMTCTOK, *PFMTCTOK;
```

Figure 10. Layout of the DD&C Condition Token

Figure 10 shows the structure of the condition token as a C-language type definition.

The fields of the DD&C condition token have the following meanings:

Condition_ID A 4-byte identifier that, together with the **uchFacility_ID**, describes the processing condition.

Note: The alphanumeric character conversion routines supplied by CDRA use a different definition of the condition token. Refer to “CDRA Return Codes” on page 115.

fCase A 2-bit field that defines the format of the **Condition_ID** portion of the token. In DD&C, its value is always **Case 1 - Service Condition**. identifiers are used. Service condition identifiers consist of the 2-byte message severity,**usMsgSev**, and 2-byte message number, **usMsgNo**.

fSeverity A 3-bit field that indicates the severity of the condition. Severity values are the same as defined for a “Case 1” **Condition_ID**.

fControl A 3-bit field that contains flags that describe or control various aspects of condition handling. In DD&C, this field has the binary value B'001' to indicate that the **uchFacility_ID** is assigned by IBM.

uchFacility_ID A 3-character alphanumeric string that identifies a product or component of a product. The combination of **uchFacility_ID** and **usMsgNo** fields uniquely identifies a condition.

The string **FMT** is used for DD&C and all its components.

pI_S_Info Instance-specific information. You access this information by using a C-language UNION construct.

For **data-type conversion routines**, it is a 32-bit ADL exception identifier that identifies the error that has occurred. “ADL Exceptions” on page 112 contains a complete list of possible ADL exceptions.

For the **Conversion Plan Builder**, it is a pointer to the FMTISINF union, which is the ADL communications area, FMTADLCA.

For the **Conversion Plan Executor**, it is a pointer the FMTISINF union. The FMTISINF union contains the ADL communication area.

However, if you used an ADL Plan CALL statement to call a user-exit which returned a non-zero severity code, and the Conversion Plan Executor returns either:

> CPX_USEREXIT_WARNING, or
> CPX_USEREXIT_ERROR

the FMTISINF union contains a copy of the Condition Token of the user-provided function.

The ADL Communications Area

The ADL Communications Area, FMTADLCA, contains information that is returned to the caller of a conversion plan when an exception occurs. The format of the ADL communications area is shown in Figure 11.

```
typedef struct _CHARPRE
{
  USHORT usLength;
  UCHAR  uchData[255];
} CHARPRE;

typedef struct _FMTADLCA
{
   LONG          lLength;
   LONG          lExId;
   USHORT        usSevCod;
   CHARPRE       PlanId;
   LONG          lPlanStmt;
   CHARPRE       InpErrDta;
   CHARPRE       SrcFldId;
   CHARPRE       TrgFldId;
} FMTADLCA, *PFMTADLCA;
```

Figure 11. Layout of the ADL Communications Area

The CHARPRE data type is provided for compatibility with the ADL definition of a string prefixed by its length. It corresponds to the following ADL declaration:

```
CHARPRE PRELEN(16) PRESIGNED(FALSE) PREBYTRVS(FALSE) MAXLEN(255)
        UNITLEN(8) MAXALC(TRUE)
```

The fields of the ADL Communications Area have the following meanings:

lLength Specifies the length of the ADL Communications Area structure.
lExId Specifies an ADL exception identifier (see “ADL Exceptions” on page 112).

usSevCod Specifies the severity of the ADL exception:

Value	**Meaning**
0	INFORMATION.
1	WARNING. Service completed, probably correctly.
2	ERROR detected. Correction was attempted. Service completed, but possibly incorrectly.
3	SEVERE error detected. Service not completed.
4	CRITICAL error detected. Service not completed and error condition signaled.

PlanId Identifies the conversion plan being executed when the exception was detected.

lPlanStmt Specifies the number of the PLAN statement that was being executed when the exception was detected.

InpErrDta Specifies the portion of the input data being processed when the exception was detected.

SrcFldId Specifies the identifier of the source field being processed when the exception was detected.

TrgFldId Specifies the identifier of the target field being processed when the exception was detected.

The Consistency Token

The Consistency Token (CNSTKN) provides the physical composition of an ADLDCLSPC in terms of:

- The order in which ADL objects appear within the space,
- Where empty spaces exist
- Whether inaccessible objects exist within the space.

The Consistency Token is returned after a call to the FMTPRS function. It is pointed to by pADLDclSpcCNSTKN. By comparing the consistency token's returned from successive parsings of ADL source text, you can determine whether the resulting ADLDCLSPCs are identical.

The Consistency Token is shown below:

22	X'3062'	16-byte value of consistency token

Value	**Meaning**
22	This 4-byte field indicates the total length of the structure, from the beginning of this length field to the end of the Consistency Yoken.
X'3062'	This 2-byte hex value indicates that the data that follows is a Consistency Token.

Figure 12 shows the C-language type definition of the structure.

```
typedef struct _FMTCNSTKN  {
   ULONG                   ulLength;
   USHORT                  usClass;
   BYTE                    abValue[16];
} FMTCNSTKN, * PFMTCNSTKN;
```

Figure 12. Layout of the Consistency Token

Data Overflow

The DD&C alphanumeric conversion routines, CDRGESP and CDRGCTL, require that data areas or arrays be allocated by the invoking program for:

- An input parameter specifying the size of the data area that is to receive the converted data.
- An output parameter that returns the size of the data area required by the conversion routine to contain all the converted data.

Data overflow occurs when the size of the data area provided by the invoking program is smaller than the size required by the function. If this happens:

1. DD&C returns the data in blocks that are the size of the data area allocated by the invoking program.
2. DD&C returns a non-zero feedback code to indicate that there is more data.
3. The invoking program calls the function repeatedly until all the data is obtained.

For example, assume the amount of allocated space is 4, the required space is 15, and the two variables are N1 for the amount of allocated space and N2 for the amount of space actually required. The algorithm for calling this function is similar to the following pseudo-code segment:

```
Set N1 to four
Set N2 to zero
Do
  Call function(ccsid, N1, N2, space, feedback)
  Process returned data
Until N1 >= N2
```

The variable N2 acts as both an input and an output variable. Notice that N2 must be initially set to zero by the invoking program. Table 7 on page 26 shows the values of N1 and N2 passed to and returned from the function for each call.

Table 7. Example of Data Overflow

Event	N1	N2	Data Remaining
First call to routine	4	0	0
Return code	–	15	11
Second call to routine	4	15	11
Return code	–	11	7
Third call to routine	4	11	7
Return code	–	7	3
Fourth call to routine	4	7	3
Return code	–	3	–

The sequence of events is:

1. The invoking program calls the function with N2 initially set to zero.
2. If the returned value of N2 is greater than N1 and a nonzero return code indicates that there is more data, the invoking program processes the returned segment of N1 elements and calls the function again for the remaining data.
3. The function is called repeatedly for as long as the value of N2 is greater than N1. The invoking program must not change the values of N1 or N2 between function calls.
4. When the returned value of N2 is less than or equal to N1, the function has returned all the requested data, consisting of N2 valid elements. The return code returned with the last block of data is zero unless an error condition unrelated to data overflow is encountered by the function.
5. The invoking program examines the return code after each call to ensure that there are no other conditions reported by the performing function—do not rely completely on the changing values of N1 and N2 to determine when the function is successful.

Errors that may arise are:

- N2 is less than or equal to N1, but feedback is not equal to zero.
- N2 is greater than N1, but the start address of the block of data to be returned is outside the range of 1 to the maximum possible value of N2.

Specific errors are documented with the individual API function descriptions.

Chapter 3. The ADL Declaration Translator APIs

This chapter describes the ADL Declaration Translator APIs (application program interfaces):

FMTPRS This function parses the ADL source files and creates binary encodings of the data that is passed as input to the Conversion Plan Builder.

FMTGEN This function generates ADL source files from the binary encodings of data.

FMTPRS - Parse Source Text

Purpose

The FMTPRS function:

- Reads a file that contains the ADL source text
- Parses the file
- Builds ADL declare space (ADLDCLSPC) and ADL plan space (ADLPLNSPC) objects.

The FMTPRS function subsequently passes these objects as input parameters to the Conversion Plan Builder API. You can also specify that FMTPRS generate a listing file. This file contains a listing of the ADL source code and messages that describe errors that are detected by the Parse function during compilation.

Prerequisite for this function is an ADL source file that you created with a text editor. This source file must contain at least one ADL DECLARE statement or one ADL PLAN statement. See *SMARTdata UTILITIES A Data Language Reference for Data Description and Conversion* for the complete specification of ADL.

Format

```
void FMTPRS(PBYTE       pbDclXlrId,                /* INPUT  */
            FMTCCSID     lParameterCCSID,          /* INPUT  */
            LONG          lSrcFilNamLength,         /* INPUT  */
            PCHAR         pchSrcFilNam,             /* INPUT  */
            FMTCCSID     lSrcFilCCSID,             /* INPUT  */
            LONG          lDclXlrOptLength,         /* INPUT  */
            PCHAR         pchDclXlrOpt,             /* INPUT  */
            LONG          lLstOptLength,            /* INPUT  */
            PCHAR         pchLstOpt,                /* INPUT  */
            LONG          lLstFilNamLength,         /* INPUT  */
            PCHAR         pchLstFilNam,             /* INPUT  */
            LONG          lADLDclSpcLength,         /* INPUT  */
            PBYTE         pbADLDclSpc,              /* OUTPUT */
            FMTCCSID     lADLDclSpcCCSID,          /* INPUT  */
            PFMTCNSTKN   pADLDclSpcCNSTKN,         /* OUTPUT */
            LONG          lADLPlnSpcLength,         /* INPUT  */
            PBYTE         pbADLPlnSpc,              /* OUTPUT */
            PFMTCTOK     pFeedBack );              /* OUTPUT */
```

Parameters

pbDclXlrId

An input variable that contains the name of the declaration translator that is to parse the source text.

This parameter is a pointer to a 16-byte field. The first 8 bytes must contain the name of the declaration translator. DD&C only supports the ADL Declaration Translator, which has the value X'2B12 0003 0188 6D01' . In the *fmtb.h* include file shipped with DD&C, the define statement,

ADLDECLTRANSLATOR specifies this value. All other input values result in the following message:

```
The value of the pbDclXlrId parameter is incorrect.
```

DDM reserves bytes 9 to 16 inclusive. The FMTPRS function ignores these bytes.

lParameterCCSID Parameter that is reserved. The FMTPRS function ignores this parameter.

lSrcFilNamLength An input variable that contains the length of the area addressed by the **pchSrcFilNam** parameter.

pchSrcFilNam The start address of an area that contains the name of the ADL source file to be parsed.

This area can contain a fully-qualified ASCII filename[5] , or a filename only, in which case the current directory is searched for the file.

lSrcFilCCSID Parameter that is reserved. The FMTPRS function ignores this parameter.

lDclXlrOptLength An input variable that contains the length of the area addressed by the **pchDclXlrOpt** parameter.

pchDclXlrOpt The start address of an area that directs the declaration translator to use the AUTOSKIP option.

> If you set **AUTOSKIP**, The FMTPRS function automatically generates SKIP statements according to the ADL alignment rules.
>
> You must insert SKIP statements in the appropriate places in the ADL source text if:
>
> - You do **not** set **AUTOSKIP**, or
> - **pchDclXlrOpt** is null, or
> - **lDclXlrOptLength** is less than or equal to zero.
>
> The ADL Declaration Translator performs checks and returns appropriate errors if the alignment of fields is incorrect.

lLstOptLength An input variable that contains the length of the area addressed by the **pchLstOpt** parameter.

pchLstOpt The start address of an area that contains listing options. The options to use are expressed as an ASCII character string.

If **pchLstOpt** is NULL or **lLstOptLength** is less than or equal

[5] A fully-qualified ASCII file name is one that contains both the directory structure and file name. An example is \DDC2\ADLSOURCE or /DDC2/ADLSOURCE on AIX systems.

to zero, then the FMTPRS function uses the default options, and does not flag listing and information messages.

If the listing options that are used contradict each other, then the last option that is specified is used. You can specify Listing options in any order. At least one blank must separate listing options.

When the FMTPRS function parses the ADL source, it produces a listing file which contains the source listing and messages. You can control when a source listing is produced using the following options:

- **NOLIST** - No source listing is produced. NOLIST is the default if the **pchLstOpt** area does not contain any listing options. If there are other listing options in the **pchLstOpt** area, they override the NOLIST option.
- **LIST** - A source listing of the ADL source is produced. The FMTPRS function replaces any ADL INCLUDE statements in the source text with the included ADL source text. A line number precedes each source line in the listing, for example:

  ```
  <0025> b: BINARY;
  ```

  Any error messages follow the text line. These error messages describe exceptions that are detected while parsing the text line.
- **FLAG(x)** - Specifies the minimum severity level of the exceptions for which messages are to be listed. The FLAG(x) option is valid independently from the LIST option. You can specify the following FLAG options (in order of increasing severity):

I	Information
W	Warning
E	Error
S	Severe

  If you do not specify the FLAG option, the FMTPRS function assumes FLAG(I).

  Messages have the following format:

  1. The current ADL source file name
  2. The line and column number where the exception occurred
  3. The error level
  4. The exception number
  5. Text that explains the exception.

An example of the message format is:

```
source.adl(125:24) : error FMT1432: Duplicate ENUMERATION
                                    value name
```

lLstFilNamLength An input variable that contains the length of the area addressed by the **pchLstFilNam** parameter.

pchLstFilNam The start address of an area that contains the name of a listing file. You should always specify the name of a listing file. If you do not specify a path, the FMTPRS function stores the listing file in the current directory. If you specify that no listing is to be produced (NOLIST option), the FMTPRS function writes any ADL error messages that occur to this file. If you specify a file name that cannot be opened, the FMTPRS function generates the error message:

```
Error opening listing file
```

If a file with the same name already exists, the FMTPRS function overwrites the file when the function is called.

lADLDclSpcLength An input variable that contains the length of the area addressed by the **pbADLDclSpc** parameter. If you set this parameter to 0, the **pbADLDclSpc** parameter value can be NULL and the FMTPRS function does not generate an ADLDCLSPC.

If the ADL source contains a DECLARE statement, an exception informs you that the ADLDCLSPC size is too small. The value specifies the exception in the **pFeedBack** parameter. To generate an ADLDCLSPC, this parameter must have a value of at least 4 so that the FMTPRS function can return the required length for the ADLDCLSPC.

pbADLDclSpc The start address of an area in which the ADLDCLSPC is returned to the caller. When returned, the first four bytes of the declaration space indicate the length of the space, in the format of an ULONG. However, if the area that is required by the declaration space exceeds the value that is specified by **lADLDclSpcLength**, the object's length field is set to the required length. The value in **pFeedBack** indicates an exception.

If no declaration space is generated, this field is set to zero.

lADLDclSpcCCSID Parameter that is reserved. The FMTPRS function ignores this parameter.

pADLDclSpcCNSTKN A pointer to the *consistency token* (PFMTCNSTKN) structure. This structure is described in “The Consistency Token” on page 24.

The consistency token is only calculated when the ADLDCLSPC is generated successfully and **pADLDclSpcCNSTKN** is not NULL.

lADLPlnSpcLength An input variable that contains the length of the area addressed by the **pbADLPlnSpc** parameter. If this parameter is set to 0, the next parameter value can be NULL and no ADLPLNSPC is generated.

If the ADL source contains a PLAN statement, the value in **pFeedBack** indicates an exception and notifies you that the ADLPLNSPC size is too small. To generate an ADLPLNSPC, this parameter must have a value of at least 4 so that the required length for ADLPLNSPC is returned.

pbADLPlnSpc The start address of an area in which the plan space is returned to the caller. When returned, the first four bytes of the plan space indicate the length of the space, in the format of an ULONG. However, if the area that is required by the plan space exceeds the value that is specified by **lADLPlnSpcLength**, the object's length field is set to the required length. The value in **pFeedBack** indicates an exception.

If no plan space is generated, this field is set to 0.

pFeedBack The condition token is returned to the caller in an area of memory whose start address is specified by **pFeedBack**. The condition token consists of a message severity and a message number that identify the error that occurred during the processing of the function. See “The Condition Token” on page 20 for the layout and description of the condition token.

Return codes

Message severity	Message number	Mnemonic	Explanation
0	0	PRS_NO_ERROR	FMTPRS executed successfully.
2	1	PRS_ERROR	A parser error occurred during processing. See the listing file for further information.
3	2	PRS_ERR_INTERNAL	Internal error. The cause of the error is specified in the trace file.
3	3	PRS_ERR_LIST_OPEN	Error opening listing file.
3	4	PRS_ERR_LIST_WRITE	Error writing to listing file.
2	5	PRS_ERR_DCLSPC	The ADLDCLSPC area is too small or not in a valid memory location.
2	6	PRS_ERR_PLNSPC	The ADLPLNSPC area is too small or not in a valid memory location.
2	7	PRS_ERR_DCLPLNSPC	Both the ADLDCLSPC and ADLPLNSPC areas are too small or not in valid memory locations.

Message severity	Message number	Mnemonic	Explanation
3	8	PRS_ERR_XLRID	The value of the **pbDclXlrId** parameter is not supported.
3	10	PRS_ERR_LSTOPT	The value of the **pchLstOpt** or **lLstOptLength** parameter is not valid.
3	11	PRS_ERR_XLROPT	The value of the **pchDclXlrOpt** or **lDclXlrOptLength** parameter is not valid.
3	12	PRS_ERR_SOURCE	The value of the **pchSrcFilNam** or **lSrcFilNamLength** parameter is not valid.
3	13	PRS_ERR_INV_LSTOPT	An unknown listing option was specified in **pchLstOpt**. The **pl_S_Info.ulAdlExId** field contains the number of the listing option that is not recognized.
3	14	PRS_ERR_INV_XLROPT	An unknown parser option was specified in **pchDclXlrOpt**. The **pl_S_Info.ulAdlExId** field contains the number of the parser option not recognized.

Comments

1. You can call the FMTPRS function from more than one process simultaneously.
2. The DDM architecture defines the **lParameterCCSID**, **lSrcFilCCSID**, and **lAdlDclSpcCCSID** parameters. DD&C does not support these parameters. This implies that character conversion of ADL source text is not possible using this function.

FMTGEN - Generate Source Text

Purpose

The FMTGEN function generates a file that contains ADL source code from an ADL declaration space (ADLDCLSPC) or an ADL plan space (ADLPLNSPC) object. The FMTPRS function produces the ADLDCLSPC and ADLPLNSPC objects. Optionally, you can generate a listing file by using the pchLstOpt parameter.

Format

```
void FMTGEN(PBYTE       pbDclXlrId,            /* INPUT  */
            FMTCCSID    lParameterCCSID,       /* INPUT  */
            LONG        lDclXlrOptLength,      /* INPUT  */
            PCHAR       pchDclXlrOpt,          /* INPUT  */
            PBYTE       pbAdlSpc,              /* INPUT  */
            FMTCCSID    lAdlSpcCCSID,          /* INPUT  */
            LONG        lSrcFilNamLength,      /* INPUT  */
            PCHAR       pchSrcFilNam,          /* INPUT  */
            FMTCCSID    lSrcFilCCSID,          /* INPUT  */
            LONG        lLstOptLength,         /* INPUT  */
            PCHAR       pchLstOpt,             /* INPUT  */
            LONG        lLstFilNamLength,      /* INPUT  */
            PCHAR       pchLstFilNam,          /* INPUT  */
            FMTCCSID    lLstFilCCSID,          /* INPUT  */
            PFMTCTOK    pFeedBack );           /* OUTPUT */
```

Parameters

pbDclXlrId An input variable that contains the name of the declaration translator of the representation domain for which the ADL source text is to be generated. It is a pointer to a 16-byte field.

DD&C only supports the ADL Declaration Translator, which has the value X'2B12 0003 0188 6D01'. In the *fmtb.h* include file shipped with DD&C, the define **ADLDECLTRANSLATOR** specifies this value. All other input values result in the following message:

```
The value of the pbDclXlrId parameter is incorrect.
```

The DDM architecture reserves bytes 9 to 16 inclusively. The FMTGEN function ignores these bytes.

lParameterCCSID Parameter that is reserved. The FMTGEN function ignores this parameter.

lDclXlrOptLength Parameter that is reserved. The FMTGEN function ignores this parameter.

pchDclXlrOpt Parameter that is reserved. The FMTGEN function ignores this parameter.

pbAdlSpc

The start address of the area that contains the ADLDCLSPC or the ADLPLNSPC object from which ADL source code is to be generated.

The FMTPRS function creates these objects as outputs.

lAdlSpcCCSID

Parameter that is reserved. The FMTGEN function ignores this parameter.

lSrcFilNamLength

An input variable that contains the length of the area addressed by the **pchSrcFilNam** parameter.

pchSrcFilNam

The start address of an area that contains the name of the ADL source file to be generated.

This area can contain, for example, a fully-qualified ASCII filename (such as `/DDC2/ADLSOURC.ADL`), or a filename only. If the area contains a filename only, the FMTGEN function creates the file in the current directory.

If a file with the same name already exists, it is overwritten.

lSrcFilCCSID

Parameter that is reserved. The FMTGEN function ignores this parameter.

lLstOptLength

An input variable that contains the length of the area addressed by the **pchLstOpt** parameter. If the length is 0, the FMTGEN function uses the default list options.

pchLstOpt

The start address of an area that contains listing options. If the pointer is NULL, the FMTGEN function uses the default options. You can specify the options in any order. At least one blank must separate the options.

A listing file contains a symbolic representation of the ADLDCLSPC or ADLPLNSPC. The listing file also contains any messages that are produced while generating the ADL source. You can control when a source listing is produced using the following options:

- **LIST** - The listing contains a symbolic representation of the space.
- **NOLIST** - No symbolic representation is produced. You can specify either NOLIST or LIST in the list options; NOLIST is the default.
- **FLAG(x)** - Specifies the minimum severity level of the exceptions for which messages are to be listed. The FLAG(x) option can be used independently of the LIST option. You can specify the following FLAG options, in order of increasing severity:

I	Information
W	Warning
E	Error

S Severe

If you do not specify the FLAG option, this function assumes FLAG(I).

Messages have the following format:

1. The error level
2. The exception number
3. If the exception occurred within the space, the offset is relative to the beginning of the space field in the ADLDCLSPC or ADLPLNSPC.
4. Text that explains the meaning of the exception.

An example of the message format is:

```
source.adl(125:24) : error FMT2859: offset 000001E6: anonymous
                     ADLCNS in constant declaration chain
```

lLstFilNamLength An input variable that contains the length of the area addressed by the **pchLstFilNam** parameter. If the value of this parameter is zero, no listing file is produced.

pchLstFilNam The start address of an area that contains the name of a listing file, if one is to be produced. If the pointer is NULL, the FMTGEN function does not produce a listing file.

If a file with the same name already exists, this function overwrites the file.

Messages that are produced while generating the ADL source are also written to the listing file, depending on the setting of the FLAG(x) option.

lLstFilCCSID Parameter that is reserved. The FMTGEN function ignores this parameter.

pFeedBack The condition token is returned to the caller in an area of memory whose start address is specified by **pFeedBack**. The condition token consists of a message severity and a message number that identify the error that occurred during the processing of the function. See “The Condition Token” on page 20 for the layout and description of the condition token.

Return codes

Message severity	Message number	Mnemonic	Explanation
0	0	GEN_NO_ERROR	Generation finished without errors.
1	1	GEN_WARNING	Warning occurred. See list file for further information.
2	2	GEN_ERROR	Error occurred. See list file for further information.
3	3	GEN_SEV_ERROR	A severe error occurred. See the list file for further information.
3	4	GEN_ERR_INTERNAL	An internal error occurred.
3	5	GEN_ERR_MEMORY	Insufficient memory available.
3	6	GEN_ERR_LSTOPT	Incorrect listing option specified.
3	8	GEN_ERR_LIST_NLEN	The **LstFilNamLength** parameter value is incorrect.
3	9	GEN_ERR_LIST_OPEN	Error opening list file.
3	10	GEN_ERR_LST_WRITE	Error writing list file.
3	12	GEN_ERR_SRC_NLEN	The **lSrcFilNamLength** parameter value is incorrect.
3	13	GEN_ERR_SRC_OPEN	Error opening source file.
3	14	GEN_ERR_SRC_WRITE	Error writing source file.
3	15	GEN_ERR_INV_XLRID	The value of the **pDclXlrId** parameter is incorrect.
3	18	GEN_ERR_LSTOPT_LEN	The value of the **lLstOptLength** parameter is incorrect.

Comments

1. You can call the FMTGEN function from more than one process simultaneously.
2. The DDM architecture also describes the parameters **usParameterCCSID**, **lLstFilCCSID**, and **usSrcFilCCSID**. DD&C for Windows does not support these parameters, implying that character conversion of ADL source text is not possible.

Chapter 4. The Conversion Plan Builder API

This chapter describes the FMTCRCP function, which represents the Conversion Plan Builder API.

FMTCRCP - Create Conversion Plan

Purpose

The input to the Conversion Plan Builder is:

- At least one ADLDCLSPC object.
- At lease one default ADLPLNSPC object. Default ADLPLNSPCs contain the plans that are to be generated for a particular application.
- Optionally, user-defined ADLPLNSPC objects. User-defined ADLPLNSPCs can be specified to overwrite default ADLPLNSPCs with the same name.

The ADLDCLSPC objects are encodings of ADL DECLARE sections describing different views of data. The ADLPLNSPC objects are encodings of ADL PLAN sections, and specify how the data is to be converted from one view to another.

As output, the Conversion Plan Builder creates a separate conversion plan for each PLAN section found in the ADLPLNSPC objects. The Conversion Plan Builder stores the generated conversion plans in a contiguous area of memory called the *conversion plan space*. You must supply the address of this area of memory as an input to this function.

If user-defined ADLPLNSPCs are supplied for which no default ADLPLNSPC of the same name exists, a warning (CPB_NO_EQ_DEFAULT_PLAN_NAME) is issued and no conversion plan is created.

If the conversion plan or plans that are generated do not fit into the block of memory allocated, the return code CPB_CNVPLNSPC_NO_MEMORY is issued and the required buffer size is supplied in the first four bytes of the returned conversion plan space.

Format

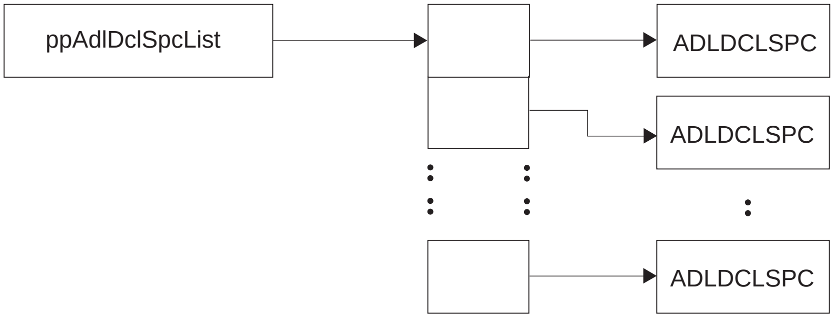
```
void FMTCRCP( ULONG    ulAdlDclSpcCount,        /* INPUT  */
              PBYTE   *ppAdlDclSpcList,         /* INPUT  */
              ULONG    ulUserAdlPlnSpcCount,    /* INPUT  */
              PBYTE   *ppUserAdlPlnSpcList,     /* INPUT  */
              ULONG    ulDefaultAdlPlnSpcCount, /* INPUT  */
              PBYTE   *ppDefaultAdlPlnSpcList,  /* INPUT  */
              ULONG    ulCnvPlnSpcLength,       /* INPUT  */
              PVOID    pCnvPlnSpc,              /* OUTPUT */
              ULONG    ulFlagList,              /* INPUT  */
              PFMTCTOK pFeedBack);              /* OUTPUT */
```

Parameters

ulAdlDclSpcCount An input variable containing the number of ADLDCLSPC addresses in the **ppAdlDclSpcList** parameter. The length of the **ppAdlDclSpcList** parameter is calculated by multiplying the result of `sizeof(PBYTE)` by the value of this parameter. It must be greater than zero.

ppAdlDclSpcList

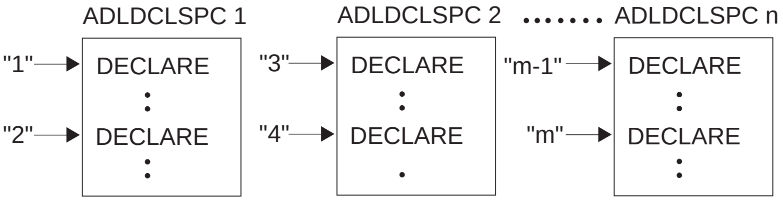
The address of an array that contains the addresses of the ADLDCLSPC objects supplied as inputs.



The first four bytes of an ADLDCLSPC indicate the length of the declaration space.

The order of the ADLDCLSPCs in the list is determined by the caller.

If the PLAN uses ADL positional identifiers to refer to the data declared in the ADLDCLSPC, the elements are numbered as they occur in the ADLDCLSPC:

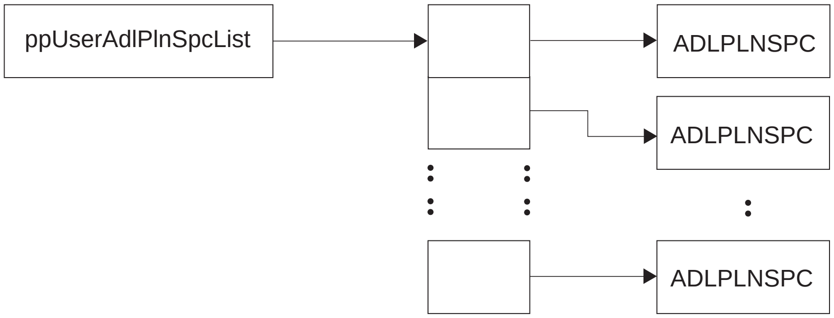


The declaration spaces must be in the format generated by the FMTPRS function of the ADL Declaration Translator. This implies that an application program cannot call the FMTCRCP function without having first called the FMTPRS function.

ulUserAdlPlnSpcCount

This input variable contains the number of user ADLPLNSPC addresses in the **ppUserAdlPlnSpcList** parameter. The total length of the **ppUserAdlPlnSpcList** parameter can therefore be calculated by multiplying the result of `sizeof(PBYTE)` by the value specified by this parameter. The value can be zero when no user defined ADLPLNSPC object is supplied.

ppUserAdlPlnSpcList The address of an array that contains the addresses of the ADLPLNSPC objects that are supplied as inputs to this function.



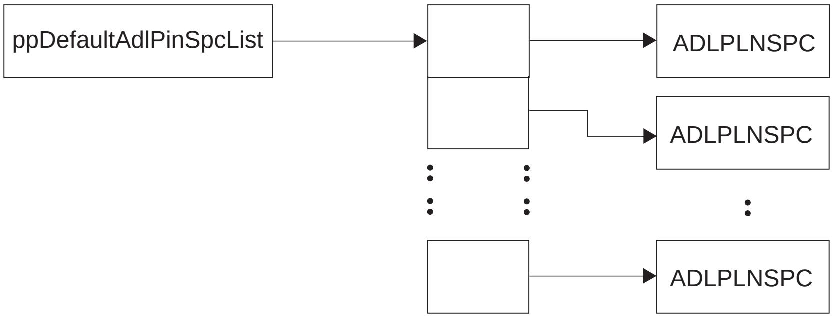
The plan spaces must be in the format generated by the FMTPRS function of the ADL Declaration Translator. This implies that an application program cannot call the FMTCRCP function without having first called the FMTPRS function.

These ADLPLNSPCs can contain plans with the same names as those in the default ADLPLNSPCs. You must decide which default plans are required and which names to use for the plans. Plans in user-defined ADLPLNSPCs are taken instead of those in default ADLPLNSPCs with the same names.

When ulUserAdlPlnSpcCount is zero, ppUserAdlPlnSpcList must also be zero.

ulDefaultAdlPlnSpcCount This input variable contains the number of default ADLPLNSPC addresses in the **ppDefaultAdlPlnSpcList** parameter. The length of the **ppDefaultAdlPlnSpcList** parameter can therefore be calculated by multiplying the result of `sizeof(PBYTE)` by the value of this parameter. The value must be greater than zero.

ppDefaultAdlPlnSpcList The address of an array that contains the addresses of the default ADLPLNSPC objects which are supplied as inputs.



These ADLPLNSPCs contain default conversion plans. These plans are used to build the conversion plan space unless user-defined ADLPLNSPCs with the same name are also supplied as inputs.

The plan spaces must be in the format generated by the FMTPRS function of the ADL Declaration Translator. This implies that an application program cannot call the FMTCRCP function without first having called the FMTPRS function.

ulCnvPlnSpcLength
This input variable specifies the length of the area of memory specified with the **pCnvPlnSpc** parameter.

pCnvPlnSpc
The start address of the *conversion plan space*. The length of the conversion plan space is specified by the **ulCnvPlnSpcLength** parameter. Conversion plans generated by the Conversion Plan Builder are written to this area. The first four bytes of the returned buffer contain the actual length of the area. If the size specified with **ulCnvPlnSpcLength** was not sufficient, the required length is returned in the first four bytes (ULONG) of the conversion plan space.

Note: The conversion plan space must be at least 32 bytes long.

ulFlagList
Contains:

Bit **Meaning**

1-31 Reserved flags.

0 DDC_REDUCED_SOURCE.

Specifies whether a reduced SEQUENCE declaration is allowed.

DDC_REDUCED_SOURCE(FALSE) indicates that when a SEQUENCE declaration in the target DECLARE section contains a field which does not occur in the corresponding

SEQUENCE declaration in the source DECLARE section, the command is rejected with the ADL exception “23.—Sequence element not found”.

DDC_REDUCED_SOURCE(TRUE) indicates that a reduced SEQUENCE declaration in the source DECLARE section is allowed.

pFeedBack

The start address of an area of memory in which the condition token is returned to the caller. The condition token is made up of a message severity and a message number identifying the error that occurred during the processing of the function.

The **pI_S_Info** field of the condition token must be initialized with a pointer to the ADL communications area (FMTADLCA). Otherwise, the error message CPB_ERROR_PARAMETERS occurs.

See “The Condition Token” on page 20 for the layout and description of the condition token.

Return codes

Message severity	Message number	Mnemonic	Explanation
0	0	CPB_NO_ERROR	No error.
1	1	CPB_NO_EQ_DEFAULT_PLAN_NAME	No default plan with the same name was found for a user-supplied plan. The user plan was not processed.
2	2	CPB_ADL_EXCEPTION_SEV2	**ADL No.** **ADL Exception** 18 Assignment of complex to scalar. 23 Sequence element not found.
3	3	CPB_ERROR_PARAMETERS	An error occurred in one of the function's parameters: The number of declare spaces specified (`ulAdlDclSpcCount`) is equal to zero The number of default plan spaces specified (`ulDefaultAdlPlnSpcCount`) is equal to zero The pointer to the declare space list (`ppAdlDclSpcList`) is NULL, or an entry in the list is NULL The pointer to the default plan space list (`ppDefaultAdlPlnSpcList`) is NULL, or an entry in the list is NULL The pointer to the user-defined plan space list (`ppUserAdlPlnSpcList`) is NULL, or an entry in the list is NULL and `ulUserAdlPlnSpcCount` is greater than zero The length of the conversion plan space (`ulCnvPlnSpcLength`) specified is less than 32 bytes The pointer to the conversion plan space (`pCnvPlnSpc`) is NULL.
3	4	CPB_ERROR_NO_MEMORY	Unable to allocate sufficient working storage.
3	5	CPB_INVALID_ADLDCLSPC	Error in input ADLDCLSPC: • The field length size field in the ADLDCLSPC is incorrect. • Incorrect ADLDCLSPC format. • A variable-length field that is shorter than the maximum possible field length is followed by another field. • An unknown code point was found in the declaration space. • A comparison in a WHEN statement is not possible because of different data types.

FMTCRCP

Message severity	Message number	Mnemonic	Explanation
3	6	CPB_INVALID_ADLPLNSPC	Error in input ADLPLNSPC: • More than one identical user-defined plan name was found in the user-defined plan spaces • More than one identical default plan name was found in the list of default plan spaces • An unknown code point was found in the plan space.
3	7	CPB_ERROR_SAME_DECLARE	View and base declarations are located in the same DECLARE statement.
3	8	CPB_ERROR_BUILD_PLAN	An internal error occurred.
3	9	CPB_CNVPLNSPC_NO_MEMORY	The space specified for the conversion plan space is too small.
3	10	CPB_INVALID_ATTR_VALUE	An attribute of a PLAN parameter contains a nonnumeric value.
3	11	CPB_STMT_ID_NOT_FOUND	An ADL <qualified identifier> in an assignment or CALL statement was not found in any ADLDCLSPC.
3	12	CPB_ID_NOT_FOUND	An ADL <qualified identifier> used as a referenced field was not found.
3	13	CPB_PP_NOT_FOUND	An ADL <qualified identifier> used as a plan parameter cannot be found in any ADLDCLSPC.
3	14	CPB_STMT_ID_AMBIGUOUS	An ADL <qualified identifier> found in an assignment or CALL statement is ambiguous.
3	15	CPB_ID_AMBIGUOUS	An ADL <qualified identifier> used as a referenced field is ambiguous.
3	16	CPB_PP_AMBIGUOUS	An ADL <qualified identifier> used in a plan parameter is ambiguous.
3	17	CPB_ERROR_DCL_REFERENCED	An ADL <qualified identifier> used in an ADLDCPSPC or ADLPLNSPC consists only of an identifier of a DECLARE. This is not allowed.
3	18	CPB_ADL_EXCEPTION_SEV3	**ADL No. — ADL Exception** 1 — Conversion not supported 8 — Nonconformable arrays 9 — ENUMERATION mismatch 10 — Invalid ENUMERATION value 16 — Input area too short 17 — Output area too short
3	19	CPB_ID_CONTAINED_IN_ MULTIPLE_PARAMETERS	An ADL <qualified identifier> is contained in several plan parameters of the same plan. This is not allowed.
3	20	CPB_ERROR_INTERRUPT	An interrupt occurred during a system call.

Comments

- All conversion plans specified in the list of default ADL PLAN spaces (**ppDefaultAdlPlnSpcList** parameter) are generated. If a user-defined ADL PLAN with the same name as a default ADL PLAN exists, the user-defined plan's definitions are used to generate a conversion plan.
- The FMTCRCP function can be called simultaneously from more than one process.

Chapter 5. The Conversion Plan Executor APIs

This chapter describes the Conversion Plan Executor APIs:

FMTCPXI The FMTCPXI function initializes the procedure.

FMTCPXC The FMTCPXC function carries out the conversion. This procedure can be called one or more times.

DMTCPXT The FMTCPXT function cleans up after the procedure has finished.

Call the functions of the Conversion Plan Executor API in the order they are listed above. The conversion procedure will run the most efficiently if you use separate initialization and termination calls.

FMTCPXI - Initialize Conversion Plan Executor

Purpose

Use this function to initialize the Conversion Plan Executor. You do this by passing the address of the conversion plan space created by the Conversion Plan Builder. FMTCPXI returns a conversion plan space *handle*, which must then be specified in subsequent calls to the Conversion Plan Executor to convert data.

Format

```
void FMTCPXI(PVOID    pCnvPlnSpc,          /* INPUT       */
             PULONG   pulCnvPlnSpcHdl,     /* OUTPUT      */
             PFMTCTOK pFeedBack);          /* OUTPUT      */
```

Parameters

pCnvPlnSpc An input variable containing the address of the conversion plan space returned by the FMTCRCP (create conversion plan) function.

pulCnvPlnSpcHdl An output variable containing the address where the conversion plan space handle is returned as an output by this function.

pFeedBack The start address of an area in which the condition token is returned to the caller. This condition token contains the message severity and a message number indicating the error that occurred during processing of the function. See “The Condition Token” on page 20 for the layout and data-type description of the condition token.

Return codes

Message severity	Message number	Mnemonic	Explanation
0	0	CPX_NO_ERROR	No error occurred.
3	1	CPX_ERROR_INTERRUPT	An ERROR_INTERRUPT was returned from an Windows function.
3	2	CPX_INTERNAL_ERR	Internal error occurred.
3	3	CPX_INVALID_CODE	Incorrect conversion plan statement in conversion plan space.
3	5	CPX_INVALID_ADDR	One of the following is incorrect: • The address of the conversion plan space, or length specified with the length field of the conversion plan space. • The address of the conversion plan space handle.
3	7	CPX_INVALID_CPS	Invalid conversion plan space.
3	9	CPX_NO_MEMORY	Insufficient memory available.

Message severity	Message number	Mnemonic	Explanation
3	12	CPX_RESOURCE_LIM	Resource limits reached on Windows system, when creating a semaphore, for example.
3	15	CPX_CDRA_RESOURCE_ERROR	Error in CDRA resources when loading conversion table.
3	16	CPX_LOADMODUL_ERROR	An error occurred when loading a user-provided exit, or in obtaining the address of the user-defined function. The field **pFeedBack->pI_S_Info.ulAdlExId** contains the Windows return code (errno).
3	103	CPX_ADL_EXCEPTION_SEV3	**ADL No.** **ADL Exception** 2 CCSID not supported. 3 Invalid CCSID pair. 4 Undefined CCSID.

Comments

- This function must be called to initialize a conversion plan space before a conversion plan can be executed. This initialization must be performed separately for each Windows process.
- The same conversion plan space can be initialized more than once. For each initialization, a new conversion plan space handle is returned and new resources are allocated.
- When return code CPX_ADL_EXCEPTION_SEV3 is returned, the field **pFeedBack->pI_S_Info.ulAdlExId** contains the ADL exception.

FMTCPXC - Conversion Plan Executor Convert

Purpose

Use this function to execute a specified conversion plan. The data to be converted is contained in the input parameters, and converted data is returned in the output parameters. The conversion plan is contained in a conversion plan space that was loaded by a previous call to the FMTCPXI function, and is identified by the handle returned by that function.

Format

```
void FMTCPXC(
            ULONG     ulCnvPlnSpcHdl,        /* INPUT          */
            ULONG     ulPlnNamLength,        /* INPUT          */
            PCHAR     pchAdl_Plan_Name       /* INPUT          */
            ULONG     ulInputParmNum,        /* INPUT          */
            PVOID     *ppInputData,          /* INPUT          */
            ULONG     ulOutputParmNum,       /* INPUT          */
            PVOID     *ppOutputData,         /* OUTPUT         */
            PFMTCTOK  pFeedBack);            /* OUTPUT         */
```

Parameters

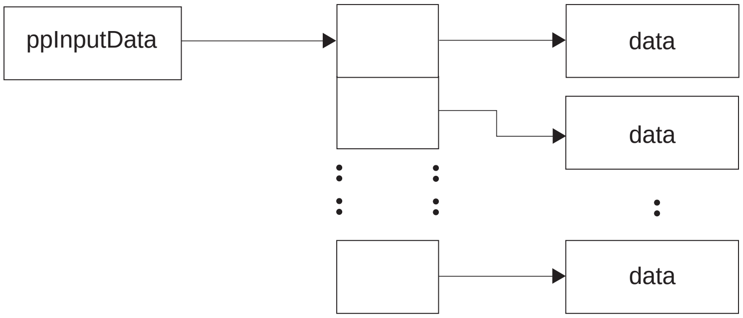
ulCnvPlnSpcHdl An input variable. This is the handle that identifies the conversion plan space containing the plans to convert. This handle is returned by the FMTCPXI function.

ulPlnNamLength An input variable that specifies the length of the conversion plan name.

pchAdl_Plan_Name An input variable that specifies the ADL identifier of the PLAN statement within the conversion plan space that is to be used for conversion.

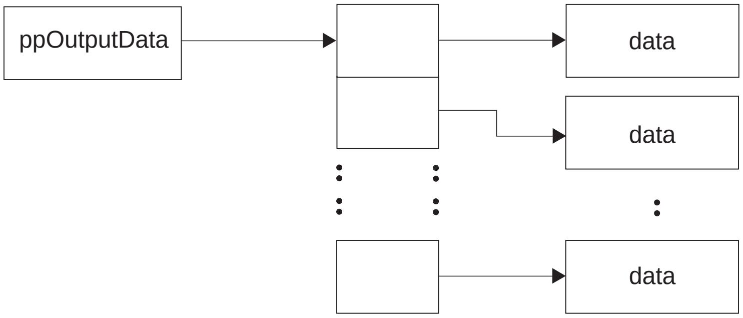
ulInputParmNum This input variable specifies the number of input parameters supplied in the **ppInputData** array. The total length of the **ppInputData** parameter can therefore be calculated by multiplying the result of `sizeof(PBYTE)` by the value specified by this parameter.

ppInputData This input variable contains the address of an array of pointers to the input parameters of the conversion plan. They must be the same number and in the same order as the original input parameters of the ADL PLAN statement.



ulOutputParmNum This input variable specifies the number of output parameters supplied in the **ppOutputData** array. The total length of the **ppOutputData** parameter can therefore be calculated by multiplying the result of `sizeof(PBYTE)` by the value specified by this parameter.

ppOutputData This output variable contains the address of an array of pointers to the output parameters of the conversion plan. They must have the same order as the original output parameters of the ADL PLAN statement.



pFeedBack The start address of an area in which the condition token is returned to the caller. This condition token contains a message severity and message number indicating the error encountered during the processing of the function.

The **pI_S_Info** field of the condition token must be initialized with a pointer to the ADL communications area. Otherwise, the error message CPX_INVALID_ADLCA occurs. See “The Condition Token” on page 20 for layout and data-type description.

FMTCPXC

Return codes

Message severity	Message number	Mnemonic	Explanation
0	0	CPX_NO_ERROR	No error occurred.
3	1	CPX_ERROR_INTERRUPT	An ERROR_INTERRUPT was returned from a Windows function.
3	2	CPX_INTERNAL_ERR	Internal error occurred.
3	3	CPX_INVALID_CODE	Incorrect contents of conversion plan.
3	4	CPX_INVALID_HANDLE	Unknown conversion plan space handle.
3	5	CPX_INVALID_ADDR	One of the following is incorrect: • The address of **pchAdl_Plan_Name** • The address of **ppInputData** when **ulInputParmNum** is greater than zero • The address of **ppOutputData** when **ulOutputParmNum** is greater than zero.
3	6	CPX_INVALID_ADLCA	Incorrect ADL communications area address.
3	8	CPX_INVALID_PLNNM	Invalid conversion plan name.
3	9	CPX_NO_MEMORY	No memory available.
3	10	CPX_PARMNBR_DIFFERS	Number of plan parameters differs.
3	11	CPX_PLAN_NOT_FOUND	Conversion plan not found in conversion plan space.
3	12	CPX_RESOURCE_LIM	Resource limits reached on the system.
1	13	CPX_USEREXIT_WARNING	User exit has returned a warning in the **pl_S_Info** area.
3	14	CPX_USEREXIT_ERROR	User exit has returned an error in the **pl_S_Info** area.
3	15	CPX_CDRA_RESOURCE_ERROR	Error in CDRA resources when loading conversion table.
3	17	CPX_MIXED_BYTE_TRUNC	Possible incorrect truncation of mixed-byte string.
1	101	CPX_ADL_EXCEPTION_SEV1	**ADL No.** **ADL Exception** 12 Assignment of negative value to unsigned field.
2	102	CPX_ADL_EXCEPTION_SEV2	**ADL No.** **ADL Exception** 5 Floating-point overflow. 11 Fixed-point overflow. 13 Floating-point underflow. 14 Undable to convert. 15 Unable to convert infinity. 18 Assignment of complex to scalar. 19 Floating-point fit violation. 21 Fixed-point constraint violation. 22 Fixed-point fit violation. 23 Sequence element not found.

Message severity	Message number	Mnemonic	Explanation
3	103	CPX_ADL_EXCEPTION_SEV3	**ADL No.** **ADL Exception** 1 Conversion not supported. 2 CCSID not supported. 3 Invalid CCSID pair. 4 Undefined CCSID. 6 Target CASE failure. 7 Invalid WHEN clause in an assignment. 8 Nonconformable arrays. 9 ENUMERATION mismatch. 10 Invalid ENUMERATION value. 20 CASE rejected. 24 Target CASE mismatch. 25 Negative array dimension size. 26 Invalid array dimension size. 27 Invalid LENGTH value of ASIS, BIT, or BITPRE field. 28 Invalid LENGTH or HIGH LOW value of CHAR or CHARPRE field. 29 Invalid LENGTH or HIGH LOW value of CHAR or CHARPRE field. 30 Input CHARSFX field contains no suffix. 31 Output CHARSFX field might contain one or more characters matching the suffix. 32 Input is DBCS and orphan byte was found. 33 Output is DBCS and output length is odd. 34 Invalid input character field. 35 Invalid digit in PACKED input field (digit>9). 36 Invalid sign in PACKED input field. 37 Invalid digit in ZONED input field (digit>9). 38 Invalid sign in ZONED input field. 39 Invalid zone nibble in ZONED input field.

Comments

- When the function returns with CPX_USEREXIT_WARNING CPX_LOADMODULE_ERROR, or CPX_USEREXIT_ERROR, the instance-specific information in the condition token (the **pI_S_Info.ulAdlExId** field) contains the DOS return code of the error.
- For performance reasons, DD&C does not check that the pointers of the Conversion Plan Executor are correct. It is the responsibility of the caller of the function to check these, otherwise segment violation errors may occur.
- When return code CPX_ADL_EXCEPTION_SEV*x* is returned, and *x* is 1, 2, or 3, the field **pFeedBack->pI_S_Info.pAdlCommArea->lExId** contains the ADL exception. The ADL communications area contains additional information about which

assignment statement was being executed when the error occurred. For further information, refer to “The ADL Communications Area” on page 23.

FMTCPXT - Terminate Conversion Plan Executor

Purpose

Use this function to terminate the use of a conversion plan space, identified by its conversion plan space handle.

Format

```
void FMTCPXT(ULONG     ulCnvPlnSpcHdl,      /* INPUT        */
             PFMTCTOK  pFeedBack);          /* OUTPUT       */
```

Parameters

ulCnvPlnSpcHdl An input variable containing the handle of the conversion plan space that is to be terminated.

pFeedBack The start address of an area in which the condition token is returned to the caller. This condition token contains a message severity and message number indicating the error encountered during the processing of the function. See "The Condition Token" on page 20 for layout and data-type description.

Return codes

Message severity	Message number	Mnemonic	Explanation
0	0	CPX_NO_ERROR	No error.
3	1	CPX_ERROR_INTERRUPT	An interrupt was returned from a function.
3	2	CPX_INTERNAL_ERR	An internal error occurred.
3	4	CPX_INVALID_HANDLE	The conversion plan space handle specified is incorrect.
3	12	CPX_RESOURCE_LIM	Resource limits reached on the system.

Chapter 6. The Data-Type Conversion Routines

This chapter describes the data-type conversion routines of DD&C.

CDRA (Character Data Representation Architecture) provides these **alphanumeric** conversion routines:

CDRMSCI This function loads conversion resources into memory.

CDRMSCP This function converts a character string from one CCSID (coded character set identifier) to another.

CDRMSCC This function releases the resources that are obtained by CDRMSCI.

CDRGESE This function retrieves ESIDs.

CDRGESP This function retrieves frequently accessed ESIDs, character and code page elements.

CDRSMXC This function obtains the CCSID for the largest character set for a specific code page.

CDRGCTL This function returns CCSID control function definitions.

The **Numeric** conversion routines describe conversions between the following formats:

- **BINARY**
- **FLOAT**
- **PACKED**
- **ZONED**

CDRMSCI - Initialize Multiple-Step Conversion

Purpose

The CDRMSCI function loads the conversion resources that are required for character data conversion, such as conversion tables and algorithms, into memory. The handle that is returned from this function is used when calling the CDRMSCP function to perform conversion.

The CDRA resources that are used by this function are:

- The CCSID resource table
- The graphic character conversion selection table
- The graphic character conversion table.

Format

```
void CDRMSCI (CDRASRV_CCSID_T    *lCCSID1,    /* IN         */
              LONG               *lST1,       /* INPUT      */
              CDRASRV_CCSID_T    *lCCSID2,    /* INPUT      */
              LONG               *lST2,       /* INPUT      */
              LONG               *lGCCASN,    /* INPUT      */
              PLONG               pToken,     /* OUTPUT     */
              CDRASRV_FeedBack_T *pFeedBack); /* OUTPUT     */
```

Parameters

lCCSID1
This variable contains the CCSID of the character string or strings to convert. Subsequent calls to the CDRMSCP function converts a character string from one CCSID to another. A positive number in the CDRA-defined range of X'0001' to X'FEFF' (1 to 65279).

lST1
The type of input string to be converted:

Type **Explanation**

0 A graphic character string with the specified CCSID. The CDRMSCP function uses an input variable which specifies the length of the string, in bytes.

1 A graphic character string that ends with a null character.

CDRA reserves the usage of types 2 through 255. Types 2 through 255 is not supported in DD&C for Windows.

lCCSID2
A variable that contains the CCSID of the converted character string or strings. Subsequent calls to the CDRMSCP function converts a character string from one CCSID to another. A positive number in the CDRA-defined range X'0001' to X'FEFF' (1 to 65279).

lST2 The type of output string to be produced:

Type	Explanation
0	A graphic character string. The CDRMSCP function uses an input variable which specifies the length of the area to hold the converted string.
1	A graphic character string that ends with a null character.
2	A graphic character string padded with spaces up to the length as specified by the **IL2** variable (see "CDRMSCP - perform multiple-step conversion" on page 63) in subsequent calls to the CDRMSCP function.

CDRA reserves the usage of types 3 through 255. Types 3 through 255 is not supported in DD&C.

lGCCASN A variable that contains a number which indicates the conversion alternative to be selected from the graphic character conversion selection table.

Value	Meaning
0 or 1	Used to select the designated installation default conversion method and conversion table or tables. 0–The IBM Default 1–The Customer Default

CDRA defines the range between 2 and 255. This range is not supported in DD&C for Windows.

pToken The start address of a handle returned by the function. This handle is used as an input parameter for subsequent calls to CDRMSCP and a subsequent call to CDRMSCC.

Note: pToken is a pointer to an array of 8 LONGs.

pFeedBack The start address of an area in which the condition token is returned to the calling program. This condition token contains a status and reason code which indicates the error that is encountered during the processing of the function. See "The Condition Token" on page 20 for the layout and type definition of the condition token and "CDRA Return Codes" on page 115 for the meaning of the status codes for CDRA.

Return codes

Status code	Reason code	Explanation
X'0000'	X'0000'	The function completed successfully.
X'0001'	X'0001'	The requested conversion is not supported (that is, there is no entry for it in the GCCST) for the specified combination of **ICCSID1**, **IST1**, **ICCSID2**, **IST2**, and **IGCCASN** parameters.
X'0001'	X'0005'	The conversion algorithm specified by **IGCCASN** does not support the specified combination of (**ICCSID1**, **IST1**) to (**ICCSID2**, **IST2**).
X'0001'	X'0006'	The value of **IGCCASN** is zero, but an “installation default” was not found in the graphic character conversion selection table for the specified (**ICCSID1**,**IST1**) to (**ICCSID2**, **IST2**) pair.
X'0002'	X'0001'	The value of **ICCSID1** specified is zero. This value specifies that the CCSID to use for the invoking program's data must be determined by the caller from the next higher level in some hierarchy. This could mean, for example, that if the CCSID is not specified at the field level, the value specified at the record level is used. If not specified at the record level, then the value specified at the file level must be used. This value must be resolved before invoking this function.
X'0002'	X'0002'	**ICCSID2** is zero, which is reserved to indicate defaulting to a higher level in a hierarchy. The invoking program must resolve the default before invoking this function.
X'0003'	X'0001'	**ICCSID1** has one of the special-purpose CCSID values in the range 65280 to 65535.
X'0003'	X'0002'	**ICCSID2** has one of the special-purpose CCSID values in the range 65280 to 65535.
X'0005'	X'0007'	A space-padded string was specified (the **IST2** parameter is equal to 2), but either the necessary space character is not defined in the CDRA resources, or the CCSID resource table definition could not be found.
X'0006'	X'0001'	The graphic character conversion selection table was not found.
X'0006'	X'0002'	A CDRA resource is currently unavailable.
X'0006'	X'0003'	The conversion method identified in the graphic character conversion selection table for the specified selection is currently unavailable.
X'0006'	X'0004'	A conversion table identified in the graphic character conversion selection table for the specified selection is not found.
X'0006'	X'0007'	Unable to generate the requested handle (**pToken** parameter).
X'0007'	X'0001'	The structure of the graphic character conversion selection table accessed by the function is incorrect.
X'0007'	X'0002'	The structure of the graphic character conversion table accessed by the function is incorrect.
X'0007'	X'0003'	The table type of graphic character conversion table does not match the method selected from the graphic character conversion selection table.
X'0008'	X'0001'	**ICCSID1** value is outside the permitted range.
X'0008'	X'0002'	**ICCSID2** value is outside the permitted range.
X'0008'	X'0003'	**IST1** value is outside the permitted range.
X'0008'	X'0004'	**IST2** value is outside the permitted range.
X'0008'	X'0007'	**IGCCASN** value is outside the permitted range.
X'0800'	X'0002'	CDRA dynamically linked load module problem occurred during initialization.

CDRMSCP - Perform Multiple-Step Conversion

Purpose

The CDRMSCP function converts a character string from one CCSID to another. The converted character string is returned to the calling program. You must supply the token that is returned from a previous call to the CDRMSCI function to perform conversion of data from one CCSID to another.

The CDRA resources that are used by this function are:

- The graphic character conversion table.

Format

```
void CDRMSCP (PLONG               pToken,     /* INPUT      */
              PCHAR               pchS1,      /* INPUT      */
              LONG               *lL1,        /* INPUT      */
              LONG               *lL2,        /* INPUT      */
              PCHAR               pchS2,      /* OUTPUT     */
              PLONG               plL3,       /* OUTPUT     */
              PLONG               plL4,       /* OUTPUT     */
              CDRASRV_FeedBack_T *pFeedBack); /* OUTPUT  */
```

Parameters

pToken The start address of the handle returned by a previous call to the CDRMSCI function.

pchS1 The start address of the string to be converted.

lL1 A positive number whose maximum value is 999 999 999. This contains:

- The length in bytes of the string to be converted when parameter **lST1** of the CDRMSCI function was equal to zero, or
- The input buffer length when the parameter **lST1** is equal to 1 (indicating a graphic character string ending with a null character).

lL2 A positive number (whose maximum value is 999 999 999) which contains the length, in bytes, of the output buffer.

pchS2 The start address of the output buffer where the converted string is placed.

plL3 The start address of a variable which contains the length in bytes of the converted string (parameter **lST2** of CDRMSCI). If applicable, this variable includes the ending null character or padding characters.

plL4 The start address of a variable which contains the number of the byte (not the offset) in the input string that caused an error. The value of this parameter depends on the manner in which the convert function ends. The possible values are as follows:

- If the function detects an output overflow condition, **plL4** is set to the first byte of the code point. This represents the next character to be converted in the input string **pchS1**.
- If the function detects an error in the input string, **plL4** contains the byte number in the input string **pchS1**. This is the part of the string that was being processed when the error was detected.
- If the conversion is error free, a value of zero is returned in **plL4**.

pFeedBack The start address of an area in which the condition token is returned to the caller. This condition token contains a status and reason code which indicates the error that is encountered during the processing of the function. See "CDRA Return Codes" on page 115 for more information.

Return codes

Status code	Reason code	Explanation
X'0000'	X'0000'	The function completed successfully.
X'0100'	X'0001'	One or more input graphic characters were replaced with a “SUB” control specified for the output string.
X'0100'	X'0002'	One or more input graphic characters were replaced with another graphic character with CCSID specified by **ICCSID2**.
X'0004'	X'0001'	The supplied output buffer was too small for the output data. A converted string that is correctly truncated and terminated and that fits within the allocated buffer is returned in the area starting at **pchS2** with its length in bytes in **pIL3**. The value in **pIL4** is set to the first byte of the code point representing the next character to be converted in the input string **pchS1**.
X'0004'	X'0002'	The encoding scheme of **ICCSID1** is X'1301' (Host Mixed SB/DB encoding). The length value in **iL2** allocated for area **pchS2** is too small for the output data. A converted string that is correctly truncated and terminated and that fits within the allocated buffer is returned in the area starting at **pchS2** with its byte-length in **pIL3**. The value in **pIL4** is set to the first byte of the double-byte character (between SO and SI brackets) that would have been converted next in the input buffer.
X'0005'	X'0001'	A double-byte **ICCSID1** was specified, but either the parameter **ST1** is equal to zero and **IL1** is odd, or **ST1** is equal to 1 and an *orphan byte* was found.
X'0005'	X'0004'	The ESID of **CCSID1** is X'1301' and there is an odd number of bytes between the SO and SI brackets.
X'0005'	X'0005'	Parameter **ST1** is equal to 1, but a null-termination character was not found in the input buffer.
X'0005'	X'0006'	Parameter **ST2** is equal to 1. However, as a result of using the selected conversion tables, the output string contains one or more characters matching the null-termination character.
X'0005'	X'0008'	A double-byte **ICCSID2** with parameter **ST2** equal to 1 was specified, together with an odd value as the length of the output buffer (**IL2**). The convert function returns only an even number of bytes (up to a maximum of the value of **IL2** minus 1 byte), including the null-termination character in **pchS2**. The contents of the remaining locations in the output buffer are unpredictable.
X'0005'	X'0009'	A double-byte **ICCSID2** with **ST2** equal to 2 (space-padded string) was specified, together with an odd value for the length of the output buffer (**IL2**). The function returns **IL2** minus 1 byte, including the space-padded characters in **ST2**. The content of the remaining locations in the output buffer are unpredictable.
X'0005'	X'000C'	The ESID of **CCSID1** is X'1301' but a trailing SI bracket is missing.
X'0005'	X'000D'	The ESID of **ICCSID1** is X'1301', and a trailing SI code point was met without first encountering the corresponding leading SO code point. (The number of intervening code points may have been odd or even; the code points may have been treated as single-byte code points because the leading SO was missing)
X'0006'	X'0006'	The control token structure is incorrect.
X'0008'	X'0005'	**IL1** is outside the permitted range.
X'0008'	X'0006'	**IL2** is outside the permitted range.
X'0800'	X'0002'	CDRA dynamically linked load module problem occurred during initialization.

CDRMSCC - Multiple-Step Conversion Cleanup

Purpose

The CDRMSCC function releases all the allocated resources and conversion information that are associated with the token that was obtained by a previous CDRMSCI function call.

The CDRA resources that are used by this function are:

- The graphic character conversion table to be deallocated.

Format

```
void CDRMSCC (PLONG               pToken,     /* INPUT/OUTPUT     */
              CDRASRV_FeedBack_T *pFeedBack); /* OUTPUT        */
```

Parameters

pToken
Start address of the handle returned from a previous call to CDRMSCI. After a successful cleanup, the handle is filled with zeros.

pFeedBack
The start address of an area in which the condition token is returned to the caller. This condition token contains a message severity and message number which indicates the error that is encountered during the processing of the function. See “The Condition Token” on page 20 for information on the layout and type definition of the condition token and “CDRA Return Codes” on page 115 for the meaning of the status codes for CDRA.

Return codes

Status code	Reason code	Explanation
X'0000'	X'0000'	The function completed successfully.
X'0006'	X'0006'	The structure of the token is incorrect.
X'0800'	X'0002'	CDRA dynamically linked load module problem occurred during initialization.

CDRGESE - Get Encoding Scheme Element and Its Subelements

Purpose

The CDRGESE function retrieves the value of the encoding scheme identifier (ESID) and each of its subelements for a given CCSID. The format of the ESID is described in “CDRA Identifiers Used in DD&C” on page 15 and in *Character Data Representation Architecture, Level 2*.

The CDRA resources that are used by this function are:

- The CCSID resource table.

Format

```
void CDRGESE (CDRASRV_CCSID_T    *lCCSID1,    /* INPUT      */
              CDRASRV_ESID_T     *pESEL,      /* OUTPUT     */
              CDRASRV_FeedBack_T *pFeedBack); /* OUTPUT     */
```

Parameters

lCCSID1 A variable which contains the CCSID value. A positive number in the CDRA-defined range of X'0001' to X'FEFF' (1 to 65279).

pESEL The start address of a structure with four elements is returned. Each element is a positive 32-bit binary number. The elements in the structure are:

ESID	Value of ESID (4352 to 65534)
BasicEncoding	The basic encoding structure (1 to 15)
NumberOfBytes	The number of bytes indicator (1 to 15)
CodeExtension	The code extension method (0 to 254).

pFeedBack The start address of an area in which the condition token is returned to the caller. This condition token contains a status and reason code which indicates the error that is encountered during the processing of the function. See “The Condition Token” on page 20 for information on the layout and type definition of the condition token and “CDRA Return Codes” on page 115 for the meaning of the status codes for CDRA.

CDRGESE

Return codes

Status code	Reason code	Explanation
X'0000'	X'0000'	The function completed successfully.
X'0001'	X'0001'	**ICCSID1** value is not in the CCSID resource repository.
X'0002'	X'0001'	**CCSID1** is zero, which is reserved for indicating a default in a hierarchy.
X'0003'	X'0001'	**ICCSID1** has one of the special-purpose CCSID values in the range 65280 to 65535.
X'0006'	X'0001'	The CCSID resource repository was not found.
X'0006'	X'0002'	The CCSID resource repository is currently unavailable.
X'0007'	X'0001'	The structure of the system CCSID resource repository accessed by the function is incorrect.
X'0007'	X'0004'	No ES element is defined in the CCSID resource table for **ICCSID1**.
X'0008'	X'0001'	**CCSID1** value is outside permitted range.
X'0800'	X'0002'	CDRA dynamically linked load module problem occurred during initialization.

CDRGESP - Get Encoding Scheme, Character Set, and Code Page Elements

Purpose

The most frequently accessed elements of a CCSID are the ESID and elements of the character set and code page. The CDRGESP function returns the value of ESID that is associated with the CCSID and the values of character set and code page elements. The format of the ESID is described in “CDRA Identifiers Used in DD&C” on page 15 and in *Character Data Representation Architecture, Level 2*.

The CDRA resources that are used by this function are:

- The CCSID resource table.

Format

```
void CDRGESP (CDRASRV_CCSID_T    *lCCSID1,    /* IN         */
              LONG               *lN1,        /* INPUT        */
              PLONG               plN2,       /* INPUT/OUTPUT  */
              CDRASRV_ESIDA_T    *plES,       /* OUTPUT       */
              PLONG               pCSCPL,     /* OUTPUT       */
              CDRASRV_FeedBack_T *pFeedBack); /* OUTPUT       */
```

Parameters

lCCSID1 A variable that contains the CCSID value. A positive number in the CDRA-defined range X'0001' to X'FEFF' (1 to 65279).

lN1 A variable that contains the number of character set and code page elements in the output area. The start address of an array of character set and code page pairs is specified by **pCSCPL**. The calling program allocates the space. This parameter contains an even number greater than or equal to 2.

plN2 The address of a variable to contain the number of values. Each character set and code page pair counts as two values. These values are associated with **lCCSID1** and returned in **pCSCPL**. The calling program must allocate sufficient space for the character set and code page elements (**lN1**). For the first call of CDRGESP, the calling program must set **plN2** to zero. For a detailed description of the function's handling of the output and value of **plN2** returned, see “Data Overflow” on page 25.

plES The start address of an encoding scheme identifier associated with **lCCSID1**. This is a 32-bit two's complement binary number in the range 4352 to 65534.

pCSCPL The start address of an array of character set and code page pairs whose format is CS1, CP1, CS2, CP2, ..., CSn, CPn.

pFeedBack The start address of an area in which the condition token is returned to the caller. This condition token contains a status and reason code which indicates the error that is encountered during the processing of the function. See “The Condition Token” on page 20 for information on the layout and type definition of the condition token and “CDRA Return Codes” on page 115 for the meaning of the status codes for CDRA.

Return codes

Status code	Reason code	Explanation
X'0000'	X'0000'	The function completed successfully.
X'0001'	X'0001'	**ICCSID1** value is not in the CCSID resource repository.
X'0002'	X'0001'	**ICCSID1** value is zero, which is reserved for indicating a default in a hierarchy.
X'0003'	X'0001'	**ICCSID1** has one of the special-purpose CCSID values in the range 65280 to 65535.
X'0004'	X'0001'	The allocated length (value of **lN1**) for the area to contain returned values was insufficient to contain all the output data that is to be returned, see “Data Overflow” on page 25.
X'0005'	X'0002'	**plN2** is greater than **lN1** at function invocation time, however, the start of the next block of data to be returned is outside the valid range from 1 to the maximum value of **lN2**. See “Data Overflow” on page 25.
X'0005'	X'000A'	**lN2** is less than or equal to **lN1** but not 0.
X'0006'	X'0001'	The CCSID resource repository was not found.
X'0006'	X'0002'	The CCSID resource repository is currently unavailable.
X'0007'	X'0001'	The structure of system CCSID resource repository accessed by the function is incorrect.
X'0007'	X'0004'	There is no ES element definition in the CCSID resource for **ICCSID1**.
X'0007'	X'0006'	There is no definition for the CS and CP elements in the CCSID resource for **ICCSID1**.
X'0008'	X'0001'	The **ICCSID1** value is outside permitted range.
X'0008'	X'0002'	The value of **lN1** is an odd number.
X'0008'	X'0003'	**lN1** is less than 2.
X'0800'	X'0002'	CDRA dynamically linked load module problem occurred during initialization.

CDRSMXC - Get CCSID With Largest Character Set for Specified Encoding Scheme and Code Page

Purpose

The CDRSMXC function gets the CCSID of the largest character set for a specified code page. The encoding scheme parameter can also be specified to distinguish between different encoding schemes of the same code page (such as PC-Display or PC-Data). CDRSMXC only works for those CCSIDs that have only one character set and code page value associated with them. That is, CDRSMXC works for pure single or pure double-byte CCSIDs for those registered to date. The character set associated with the returned CCSID is the largest (in size) of all the character sets in the CCSID resource installed on the system.

The CDRA resources that are used by this function are:

- The CCSID resource table.

Format

```
void CDRSMXC (CDRASRV_CPGID_T    *lCPIN,      /* INPUT      */
              CDRASRV_ESIDA_T    *lESIN,      /* INPUT      */
              CDRASRV_CCSID_T    *plCCSIDR,   /* OUTPUT     */
              CDRASRV_ESIDA_T    *plESR,      /* OUTPUT     */
              CDRASRV_FeedBack_T *pFeedBack); /* OUTPUT     */
```

Parameters

lCPIN
Variable which contains the code page value. A positive number in the range X'0001' to X'FDFE' (1 to 65022). The additional range X'FDFF' to X'FFFF' is defined by CDRA. DD&C does not support the additional range.

lESIN
Variable which contains the encoding scheme:

ESIN **Meaning**

0
The calling program does not know the encoding scheme and requests the first CCSID encountered that has the specified code page and the “full” or “maximum” character set. The CCSID is found in the CCSID repository.

Other
The calling program specifies the ESID value. A 32-bit two's complement binary, positive number in the range 4352 to 65534 . Only ESIDs that have a single character set and code page pair that is associated with them are valid for this function.

plCCSIDR
The start address of a variable to contain the returned CCSID value. A positive number in the CDRA-defined range of X'0001' to X'FEFF' (1 to 65279).

pIESR The start address of a variable to contain the encoding scheme value of the returned CCSID. A 32-bit two's complement binary, positive number in the range 4352 to 65534.

pFeedBack The start address of an area in which the condition token is returned to the caller. This condition token contains a status and reason code which indicates the error that is encountered during the processing of the function. See “The Condition Token” on page 20 for information on the layout and type definition of the condition token and “CDRA Return Codes” on page 115 for the meaning of the status codes for CDRA.

Return codes

Status code	Reason code	Explanation
X'0000'	X'0000'	The function completed successfully.
X'0001'	X'0001'	No entry was found in the CCSID resource repository for the specified **ICPIN** and **IESIN** combination.
X'0001'	X'0003'	ESID specified was 0. The first CCSID encountered in the CCSID repository with the specified code and the “Full” or “Maximum” character set was returned, additional CCSIDs meeting the criteria may exist.
X'0001'	X'0009'	The ESID specified indicates that more than one pair of CS and CP (control point) are associated with it, which is not allowed for this function.
X'0002'	X'0001'	**ICPIN** value is zero.
X'0003'	X'0001'	**ICPIN** value is 65535.
X'0006'	X'0001'	The CCSID resource repository was not found.
X'0006'	X'0002'	The CCSID resource repository is currently unavailable.
X'0007'	X'0001'	The structure of the system CCSID resource repository accessed by the function is incorrect.
X'0008'	X'0001'	CP value is outside permitted range.
X'0008'	X'0009'	ESIN value is nonzero and outside permitted range.
X'0800'	X'0002'	CDRA dynamically linked load module problem occurred during initialization.

CDRGCTL - Get Control Function Definition

Purpose

The CDRGCTL function returns the control function definition that is associated with a given CCSID. Each control function definition is defined as a triplet consisting of:

- The code point value that is allocated to the requested control function definition
- The width, as a number of bytes
- The state number in which the code point is used.

For each of the possible code extension switching states[6] associated with the CCSID, there is at most one code point for a control function within a switching state. A selection parameter (SEL) is used to identify which control function definition is returned by the function.

The CDRA resources that are used by this function are:

- The CCSID resource table.

Format

```
void CDRGCTL (CDRASRV_CCSID_T    *lCCSID1,    /* IN            */
              LONG               *lSEL,       /* INPUT         */
              LONG               *lN1,        /* INPUT         */
              PLONG               plN2,       /* INPUT/OUPUT   */
              PLONG               pCTLFDF,    /* OUTPUT        */
              CDRASRV_FeedBack_T *pFeedBack); /* OUTPUT        */
```

Parameters

lCCSID1 Variable which contains the CCSID value. A positive number in the CDRA defined range X'0001' to X'FEFF' (1 to 65279).

lSEL Variable which contains the selection specification, a positive number in the range 0 to 5. If the selected control function element is available in the resource definition for CCSID1, the triplet is returned in the area starting at **pCTLFDF**. The following values are currently defined for SEL:

SEL	**Selected control characters**
0	Space
1	Substitute
2	New Line
3	Line Feed
4	Carriage Return

[6] An example is the use of two switching states, the SO (shift-out) and SI (shift-in) characters. These characters are used:

- To control access to an alternative assignment of graphic characters to code points, and
- To show whether one byte or two bytes of the data constitutes a code point.

This is used in the EBCDIC (extended binary-coded decimal interchange code) mixed single-byte and double-byte codes.

5 End of File

CDRA reserves the usage of types 6 through 255. Types 6 through 255 is not supported in DD&C for Windows.

lN1 Variable which contains the size of the allocated area starting at **pCTLFDF**. The allocated area contains the returned data. This parameter is specified as a number of elements. Each triplet is counted as 3 elements. The minimum value is 3.

plN2 Start address of a variable to contain the number of values that are returned in **pCTLFDF**. It is a positive integer and is a multiple of 3 (corresponding to each triplet in **pCTLFDF**). If no definition is found in the CCSID resource, **plN2** returns a value of 0. At the first call of CDRGCTL, the calling program must set **plN2** to zero. For a detailed description of the function's handling of the output and the value of **plN2** returned, see “Data Overflow” on page 25.

pCTLFDF The start address of an array to contain the returned definition elements. Each definition element triplet consists of a code point, its width, and the state number of the code point for each possible code extension switching state for **lCCSID1**. A zero state number in the corresponding entry in **CTLFDF** indicates an undefined element.

pFeedBack The start address of an area in which the condition token is returned to the caller. This condition token contains a status and reason code indicating the error that is encountered during the processing of the function. See “The Condition Token” on page 20 for information on the layout and type definition of the condition token and “CDRA Return Codes” on page 115 for the meaning of the status codes for CDRA.

Return codes

Status code	Reason code	Explanation
X'0000'	X'0000'	The function completed successfully.
X'0001'	X'0001'	**ICCSID1** value is not in the CCSID resource repository.
X'0001'	X'0004'	One or more of the requested control function definitions are undefined (as indicated by a zero value for its corresponding state number in **pCTLFDF**).
X'0001'	X'000A'	The requested control function definition element in the CCSID resource for **ICCSID1** was not found.
X'0002'	X'0001'	**ICCSID1** value is zero, which is reserved for indicating a default in a hierarchy.
X'0003'	X'0001'	**CCSID1** has one of the special-purpose CCSID values in the range 65280 to 65535.
X'0004'	X'0001'	The allocated length (value of **lN1**) for the area to contain returned values was insufficient to contain all the output data that is to be returned, see "Data Overflow" on page 25.
X'0005'	X'0002'	**plN2** is greater than **lN1** at function invocation, but the start of the next block of data to be returned is outside the valid range from 1 to the maximum value of **lN2**. See "Data Overflow" on page 25 for details.
X'0005'	X'0003'	The value specified in the **SEL** parameter is not supported.
X'0005'	X'000A'	**lN2** is less than or equal to **lN1** but is not 0.
X'0006'	X'0001'	The CCSID resource repository was not found.
X'0006'	X'0002'	The CCSID resource repository is currently unavailable.
X'0007'	X'0001'	The structure of the system CCSID resource repository accessed by the function is incorrect.
X'0008'	X'0001'	**ICCSID1** value is outside permitted range.
X'0008'	X'0002'	Reserved.
X'0008'	X'0003'	The value of **lN1** is less than 3.
X'0008'	X'000B'	**SEL** value is outside permitted range.
X'0800'	X'0002'	CDRA dynamically linked load module problem occurred during initialization.

Numeric Conversion Routines

DD&C defines an API call for each supported numeric conversion combination. The format of the API calls is:

```
FMTxxyy (INATTR,                         /* INPUT           */
         INBUF,                          /* INPUT           */
         OUTATTR,                        /* INPUT           */
         OUTBUF,                         /* OUTPUT          */
         FeedBack);                      /* OUTPUT          */
```

where *xx* and *yy* denote two-character codes. The codes represent ADL data types of input and output fields, respectively. Codes that are supported by DD&C are:

Code	ADL data type
BN	BINARY
FL	FLOAT
PK	PACKED
ZN	ZONED

For example, FMTBNFL is the name of the BINARY-to-FLOAT conversion routine.

Each conversion routine has the following parameters:

INATTR An array which contains attribute values of the input data. The size and contents of the array depend on the data type of the input data. Each array element is a 32-bit two's complement binary number. See “Attribute Array Formats for Numeric Conversion Routines” on page 77 for a more detailed description of the array for each data type supported.

INBUF The start address of the area that contains the input data, that is, the data to be converted.

OUTATTR An array which contains attribute values of the output data. The size and the contents of the array depend on the data type of the output data. See “Attribute Array Formats for Numeric Conversion Routines” on page 77 for a more detailed description of the array for each data type supported.

OUTBUF The start address of an area in which the converted data is to be returned.

If an error occurs during conversion, the contents of this area are unpredictable.

FeedBack The start address of an area in which the condition token is returned to the caller. This condition token contains the message severity and message number which indicates the error that is encountered during the processing of the function. See “The Condition Token” on page 20 for the layout and data type description.

For performance reasons, DD&C does not check the pointers that are used as inputs to the numeric conversion routines. You should ensure that all pointers that are passed to the routines are valid, otherwise access violation errors may occur.

Attribute Array Formats for Numeric Conversion Routines

Attributes are the parameters for the numeric data-type conversion routines. The attributes are defined as a “union” whose contents can be either an array or a structure. This is necessary because not all programming languages support the concept of a structure. The size of an attribute union is fixed for a given ADL data type. Each element of the structure or array is a 32-bit two's complement binary number.

For further information about the meaning of the ADL attributes, refer to *SMARTdata UTILITIES A Data Language Reference for Data Description and Conversion*.

Attributes for BINARY

This union consists of an array and a structure with 10 elements. This union contains the values of the following ADL attributes in the order that is specified:

```
 BYTRVS
 COMPLEX
 CONSTRAINED
 FIT
 LENGTH
 PRECISION
 RADIX
 SCALE
 SGNCNV
 SIGNED
```

The union has the following declaration:

```
typdef union
{
  LONG BNAttrVector[10];
  struct
  {
    LONG  lBYTRVS;
    LONG  lCOMPLEX;
    LONG  lCONSTRAINED;
    LONG  lFIT;
    LONG  lLENGTH;
    LONG  lPRECISION;
    LONG  lRADIX;
    LONG  lSCALE;
    LONG  lSGNCNV;
    LONG  lSIGNED;
  }BNAttrRecord;
} BNATTR, *PBNATTR;
```

lBYTRVS

This is a Boolean value. It specifies whether the field is encoded in byte-reversed order:

TRUE Field is byte-reversed.
FALSE Field is not byte-reversed.

lCOMPLEX

This is a Boolean value. It indicates whether the field consists of two adjacent fields with the same attributes. The first field represents the “real” part of a complex number and the second field represents the “imaginary” part of a complex number.

TRUE Field represents a complex number
FALSE Field does not represent a complex number.

lCONSTRAINED

This is a Boolean value. It indicates whether the values that are assigned to a field must be constrained to the range that is implied by following attributes:

- RADIX
- SCALE
- PRECISION

TRUE Field is constrained
FALSE Field is not constrained.

lFIT

This assigns a numeric value to the BINARY field.

Value **Definition**

0 ROUND. The least-significant binary digits are rounded to fit.

1 TRUNCATE. The least-significant binary digits are truncated to fit.

2 EXACT. No loss of the least-significant binary digits.

lLENGTH

This specifies the length of the BINARY field in bits. The valid range is from 1 to 32. If a length of 0 is specified, the length is determined from the precision. See the rules relating to the BINARY data type in *SMARTdata UTILITIES A Data Language Reference for Data Description and Conversion* for more information.

If BYTRVS(TRUE) or COMPLEX(TRUE), the value must be a multiple of 8.

lPRECISION

This specifies the maximum number of significant binary or decimal digits and depends on the RADIX attribute. The valid range is from 1 to 32, depending on the LENGTH, RADIX, and SIGNED attributes.

lRADIX This specifies the number system base that is assumed by the SCALE and PRECISION attributes.

Value	Meaning
2	The SCALE and PRECISION attributes refer to binary digits (bits).
10	The SCALE and PRECISION attributes refer to decimal digits though the number is stored in binary format.

lSCALE This is the scaling factor of the BINARY field in the range -128 to +127.

lSGNCNV This specifies how the sign of the BINARY field is to be determined.

Value	Meaning
0	ALGEBRAIC. Assignment of negative to unsigned field is not allowed.
1	LOGICAL. Assignment of negative to unsigned field is allowed.

lSIGNED This is a Boolean value that specifies whether the field has a sign:

TRUE	Field has a sign.
FALSE	Field is unsigned.

Attributes for FLOAT

This union consists of an array and a structure with 6 elements. This union contains the values of the following attributes in the order that is specified:

```
BYTRVS
COMPLEX
FIT
FORM
PRECISION
RADIX
```

The union has the following declaration:

```
typdef union
{
  LONG FLAttrVector[6];
  struct
  {
    LONG  lBYTRVS;
    LONG  lCOMPLEX;
    LONG  lFIT;
    LONG  lFORM;
    LONG  lPRECISION;
    LONG  lRADIX;
  }FLAttrRecord;
} FLATTR, *PFLATTR;
```

IBYTRVS Boolean value. Specifies whether the field is encoded in byte-reversed order.

TRUE	Field is byte-reversed
FALSE	Field is not byte-reversed.

ICOMPLEX This is a Boolean value. This value indicates whether the field consists of two adjacent fields with the same attributes. The first field represents the “real” part of the complex number and the second field represents the “imaginary” part of the complex number.

TRUE	Field represents a complex number
FALSE	Field does not represent a complex number.

IFIT This assigns a numeric value to the FLOAT field.

Value	**Meaning**
0	ROUND. The least significant binary digits are rounded to fit.
1	TRUNCATE. The least significant binary digits are truncated to fit.
2	EXACT. No loss of the least-significant binary digits.

IFORM This specifies the form of the floating point number.

Value	**Meaning**
0	FB32. Single-precision binary floating-point number
1	FB64. Double-precision binary floating-point number
2	FB80. Extended-precision binary floating-point number
3	FH32. Single-precision hexadecimal floating-point number
4	FH64. Double-precision hexadecimal floating-point number
5	FH128. Extended-precision hexadecimal floating-point number
6	FI128. Extended-precision binary floating-point number.

IPRECISION This specifies the maximum number of significant binary or decimal digits, depending on the RADIX attribute. The valid range depends on the FORM and RADIX attributes. The PRECISION attribute has no influence on the format or value of the FLOAT field.

IRADIX This specifies the number system base that is assumed by the PRECISION attribute.

Value	**Meaning**
2	The PRECISION attribute refers to binary digits (bits).
10	The PRECISION attribute refers to decimal digits.

Attributes for PACKED

This union consists of an array and a structure with 10 elements. This union contains the values of the following attributes in the order that is specified:

```
 COMPLEX
 CONSTRAINED
 FIT
 PRECISION
 SCALE
 SGNLOC
 SGNMNS
 SGNPLS
 SGNUNS
 SIGNED
```

The union has the following declaration:

```
typdef union
{
  LONG PKAttrVector[10];
  struct
  {
    LONG  lCOMPLEX;
    LONG  lCONSTRAINED;
    LONG  lFIT;
    LONG  lPRECISION;
    LONG  lSCALE;
    LONG  lSGNLOC;
    LONG  lSGNMNS;
    LONG  lSGNPLS;
    LONG  lSGNUNS;
    LONG  lSIGNED;
  }PKAttrRecord;
} PKATTR, *PPKATTR;
```

lCOMPLEX This is a Boolean value. This value indicates whether the field consists of two adjacent fields with the same attributes. The first field represents the “real” part of the complex number and the second field represents the “imaginary” part of the complex number.

TRUE Field represents a complex number
FALSE Field does not represent a complex number.

lCONSTRAINED This is a Boolean value. This value indicates whether the values that are assigned to a field must be constrained to the range implied by the SCALE and PRECISION attributes:

TRUE Field is constrained
FALSE Field is not constrained.

lFIT

This assigns a numeric value to the PACKED field.

Value	Meaning
0	ROUND. The least-significant decimal digits are rounded to fit.
1	TRUNCATE. The least-significant decimal digits are truncated to fit.
2	EXACT. No loss of the least-significant decimal digits.

lPRECISION

This specifies the maximum number of significant decimal digits. The valid range is 1 to 31.

lSCALE

This is a scaling factor for the PACKED field in the range -128 to +127.

lSGNLOC

This specifies the location of the sign in the PACKED field. The variable is ignored if SIGNED(FALSE).

Value	Meaning
0	DGTLSTBYT. Last nibble of last byte.

lSGNMNS

This is a string of 8 hexadecimal digits and represents the sign of a negative number. The first digit represents the preferred sign, and sets the sign of the target field. If you use less than 8 hexadecimal digits, fill the remainder of the string with one of the digits that is used. See “Example” on page 84 for further information.

lSGNPLS

This is a string of 8 hexadecimal digits that is used to represent the sign of a positive number. The first digit represents the preferred sign, and sets the sign of the target field. If you use less than 8 hexadecimal digits, fill the remainder of the string with one of the digits that is used. See “Example” on page 84 for further information.

lSGNUNS

This is a string of 8 hexadecimal digits that are used to indicate an unsigned number with a sign position. The first digit sets the sign of the target field.

If you use this attribute, set lSGNMNS and lSGNPLS to zero.

If you use less than 8 hexadecimal digits, fill the remainder of the string with one of the digits that are used.

lSIGNED

This is a Boolean value that specifies whether the field includes a sign position in its representation:

TRUE	Field has a sign.
FALSE	Field does not have a sign.

Attributes for ZONED

This union consists of an array and a structure with 11 elements. This union contains the values of the following attributes in the order that is specified:

```
CCSID
COMPLEX
CONSTRAINED
FIT
PRECISION
SCALE
SGNLOC
SGNMNS
SGNPLS
SIGNED
ZONENC
```

The union has the following declaration:

```
typdef union
{
  LONG ZNAttrVector[11];
  struct
  {
    LONG  lCCSID;
    LONG  lCOMPLEX;
    LONG  lCONSTRAINED;
    LONG  lFIT;
    LONG  lPRECISION;
    LONG  lSCALE;
    LONG  lSGNLOC;
    LONG  lSGNMNS;
    LONG  lSGNPLS;
    LONG  lSIGNED;
    LONG  lZONENC;
  }ZNAttrRecord;
} ZNATTR, *PZNATTR;
```

lCCSID This specifies the CCSID of the sign if SGNLOC(FRSBYT, LSTBYT) is specified. The value has to be in the range X'1' to X'DFFF' (1 to 57343).

lCOMPLEX This is a Boolean value. This value specifies whether the field consists of two adjacent fields with the same attributes. The first field represents the “real” part of the complex number and the second field represents the “imaginary” part of the complex number.

TRUE Field represents a complex number.
FALSE Field does not represent a complex number.

lCONSTRAINED This is a Boolean value. This value specifies whether the values that can be assigned to a field must be constrained to the range implied by the SCALE and PRECISION attributes:

TRUE Field is constrained
FALSE Field is not constrained.

IFIT Assigns a numeric value to the ZONED field.

Value	Meaning
0	ROUND. The least-significant decimal digits are rounded to fit.
1	TRUNCATE. The least-significant decimal digits are truncated to fit.
2	EXACT. No loss of the least-significant decimal digits.

IPRECISION This specifies the maximum number of significant decimal digits. The valid range is 1 to 31.

ISCALE That is a scaling factor for the ZONED field in the range -128 to +127.

ISGNLOC This specifies the location of the sign in the ZONED field. The variable is ignored if SIGNED(FALSE).

Value	Meaning
1	ZONFRSBYT. First nibble of first byte.
2	ZONLSTBYT. First nibble of the last byte.
3	FRSBYT. First byte which contains the plus (+) or minus (-) character in the specified CCSID.
4	LSTBYT. Last byte which contains the plus (+) or minus (-) character in the specified CCSID.

ISGNMNS This is a string of 8 hexadecimal digits. ISGNMNS is used to represent the sign of a negative number if SGNLOC(ZONFRSBYT) or SGNLOC(ZONLSTBYT) is specified. The first digit represents the preferred sign, and is used to set the sign of the target field. If you use less than 8 hexadecimal digits, fill the remainder of the string with one of the digits that are used.

ISGNPLS This is a string of 8 hexadecimal digits. ISGNPLS is used to represent the sign of a positive number if SGNLOC(ZONFRSBYT) or SGNLOC(ZONLSTBYT) is specified. The first digit represents the preferred sign. This digit is used to set the sign of the target field. If you use less than 8 hexadecimal digits, fill the remainder of the string with one of the digits that is used.

Example:

Assume that any of the hexadecimal digits `X'C'`, `X'A'`, `X'F'`, or `X'E'` can be used to represent a plus sign (+). `X'C'` must be used as the target of conversion. The hexadecimal string to define in this case is:

```
X'CAFEEEEE'
```

To define this as a LONG constant, specify:

```
0xCAFEEEEE
```

Note: The principle that is illustrated by this example also applies to the lSGNMNS and lSIGNED attributes.

lSIGNED This is a Boolean value that specifies whether the field includes a sign position in its representation:

TRUE Field has a sign.
FALSE Field does not have a sign.

lZONENC This is the value of the zoned portion of the ZONED field. The range of valid values is `X'0'` to `X'F'`.

FMTBNBN - Binary to Binary

Purpose

The FMTBNBN function converts an input data field of the ADL BINARY data type to an output field of the ADL BINARY data type.

Format

```
void FMTBNBN(PBNATTR  pINATTR,                /* INPUT      */
             PVOID    pINBUF,                 /* INPUT      */
             PBNATTR  pOUTATTR,               /* INPUT      */
             PVOID    pOUTBUF,                /* OUTPUT     */
             PFMTCTOK pFeedBack);             /* OUTPUT     */
```

For the definition of the data type PBNATTR see “Attributes for BINARY” on page 77.

Return codes

Message severity	Message number	Mnemonic	Explanation
0	0	LCF_NO_ERROR	No error.
1	101	LCF_ADL_EXCEPTION_SEV1	**ADL No.** **ADL Exception** 12 Assignment of negative value to unsigned field.
2	102	LCF_ADL_EXCEPTION_SEV2	**ADL No.** **ADL Exception** 11 Fixed-point overflow. 18 Assignment of complex to scalar. 21 Fixed-point constraint violation. 22 Fixed-point fit violation.
3	51	LCF_INV_IN_LENGTH	LENGTH attribute of input field is invalid.
3	52	LCF_INV_IN_PRECISION	PRECISION attribute of input field is invalid.
3	53	LCF_INV_IN_RADIX	RADIX attribute of input field is invalid.
3	54	LCF_INV_IN_SCALE	SCALE attribute of input field is invalid.
3	57	LCF_INV_IN_LENGTH_BYTRVS	LENGTH of input field not multiple of 8 and BYTRVS(TRUE).
3	58	LCF_INV_OUT_FIT	FIT attribute of output field is invalid.
3	60	LCF_INV_OUT_LENGTH	LENGTH attribute of output field is invalid.
3	61	LCF_INV_OUT_PRECISION	PRECISION attribute of output field is invalid.
3	62	LCF_INV_OUT_RADIX	RADIX attribute of output field is invalid.
3	63	LCF_INV_OUT_SCALE	SCALE attribute of output field is invalid.
3	64	LCF_INV_OUT_SGNCNV	SGNCNV attribute of output field is invalid.
3	67	LCF_INV_OUT_LENGTH_BYTRVS	LENGTH of input field not multiple of 8 and BYTRVS(TRUE).

FMTBNFL - Binary to Float

Purpose

The FMTBNFL function converts an input data field of the ADL BINARY data type to an output field of the ADL FLOAT data type.

Format

```
void FMTBNFL(PBNATTR  pINATTR,               /* INPUT      */
             PVOID    pINBUF,                /* INPUT      */
             PFLATTR  pOUTATTR,              /* INPUT      */
             PVOID    pOUTBUF,               /* OUTPUT     */
             PFMTCTOK pFeedBack);            /* OUTPUT     */
```

For the attribute list for PBNATTR, see “Attributes for BINARY” on page 77. For the attribute list for PFLATTR, see “Attributes for FLOAT” on page 79.

Return codes

Message severity	**Message number**	**Mnemonic**	**Explanation**
0	0	LCF_NO_ERROR	No error.
2	102	LCF_ADL_EXCEPTION_SEV2	**ADL No.** **ADL Exception** 5 Floating-point overflow. 13 Floating-point underflow. 18 Assignment of complex to scalar. 19 Floating-point fit violation.
3	51	LCF_INV_IN_LENGTH	LENGTH attribute of input field is invalid.
3	52	LCF_INV_IN_PRECISION	PRECISION attribute of input field is invalid.
3	53	LCF_INV_IN_RADIX	RADIX attribute of input field is invalid.
3	54	LCF_INV_IN_SCALE	SCALE attribute of input field is invalid.
3	57	LCF_INV_IN_LENGTH_BYTRVS	LENGTH of input field not multiple of 8 and BYTRVS(TRUE).
3	58	LCF_INV_OUT_FIT	FIT attribute of output field is invalid.
3	59	LCF_INV_OUT_FORM	FORM attribute of output field is invalid.

FMTBNPK - Binary to Packed

Purpose

The FMRBNPK function converts an input data field of the ADL BINARY data type to an output field of the ADL PACKED data type.

Format

```
void FMTBNPK(PBNATTR  pINATTR,              /* INPUT      */
             PVOID    pINBUF,               /* INPUT      */
             PPKATTR  pOUTATTR,             /* INPUT      */
             PVOID    pOUTBUF,              /* OUTPUT     */
             PFMTCTOK pFeedBack);           /* OUTPUT     */
```

For the attribute list for PBNATTR, see "Attributes for BINARY" on page 77. For the attribute list for PPKATTR, see "Attributes for PACKED" on page 81.

Return codes

Message severity	Message Number	Mnemonic	Explanation
0	0	LCF_NO_ERROR	No error.
1	101	LCF_ADL_EXCEPTION_SEV1	**ADL No.** **ADL Exception** 12 Assignment of negative value to unsigned field.
2	102	LCF_ADL_EXCEPTION_SEV2	**ADL No.** **ADL Exception** 11 Fixed-point overflow. 18 Assignment of complex to scalar. 21 Fixed-point constraint violation. 22 Fixed-point fit violation.
3	51	LCF_INV_IN_LENGTH	LENGTH attribute of input field is invalid.
3	52	LCF_INV_IN_PRECISION	PRECISION attribute of input field is invalid.
3	53	LCF_INV_IN_RADIX	RADIX attribute of input field is invalid.
3	54	LCF_INV_IN_SCALE	SCALE attribute of input field is invalid.
3	57	LCF_INV_IN_LENGTH_BYTRVS	LENGTH of input field not multiple of 8 and BYTRVS(TRUE).
3	58	LCF_INV_OUT_FIT	FIT attribute of output field is invalid.
3	61	LCF_INV_OUT_PRECISION	PRECISION attribute of output field is invalid.
3	63	LCF_INV_OUT_SCALE	SCALE attribute of output field is invalid.

FMTBNZN - Binary to Zoned

Purpose

The FMTBNZN function converts an input data field of the ADL BINARY data type to an output field of the ADL ZONED data type.

Format

```
void FMTBNZN(PBNATTR  pINATTR,                /* INPUT      */
             PVOID    pINBUF,                 /* INPUT      */
             PZNATTR  pOUTATTR,               /* INPUT      */
             PVOID    pOUTBUF,                /* OUTPUT     */
             PFMTCTOK pFeedBack);             /* OUTPUT     */
```

For the attribute list for PBNATTR, see “Attributes for BINARY” on page 77. For the attribute list for PZNATTR, see “Attributes for ZONED” on page 82.

Return codes

Message severity	Message Number	Mnemonic	Explanation
0	0	LCF_NO_ERROR	No error.
1	101	LCF_ADL_EXCEPTION_SEV1	**ADL No.** **ADL Exception** 12 Assignment of negative value to unsigned field.
2	102	LCF_ADL_EXCEPTION_SEV2	**ADL No.** **ADL Exception** 11 Fixed-point overflow. 18 Assignment of complex to scalar. 21 Fixed-point constraint violation. 22 Fixed-point fit violation.
3	103	LCF_ADL_EXCEPTION_SEV3	**ADL No.** **ADL Exception** 2 CCSID not supported.
3	51	LCF_INV_IN_LENGTH	LENGTH attribute of input field is invalid.
3	52	LCF_INV_IN_PRECISION	PRECISION attribute of input field is invalid.
3	53	LCF_INV_IN_RADIX	RADIX attribute of input field is invalid.
3	54	LCF_INV_IN_SCALE	SCALE attribute of input field is invalid.
3	57	LCF_INV_IN_LENGTH_BYTRVS	LENGTH of input field not multiple of 8 and BYTRVS(TRUE).
3	58	LCF_INV_OUT_FIT	FIT attribute of output field is invalid.
3	61	LCF_INV_OUT_PRECISION	PRECISION attribute of output field is invalid.
3	63	LCF_INV_OUT_SCALE	SCALE attribute of output field is invalid.
3	65	LCF_INV_OUT_SGNLOC	SGNLOC attribute of output field is invalid.
3	66	LCF_INV_OUT_ZONENC	ZONENC attribute of output field is invalid.

FMTFLBN - Float to Binary

Purpose

The FMTFLBN function converts an input data field of the ADL FLOAT data type to an output field for the ADL BINARY data type.

Format

```
void FMTFLBN(PFLATTR  pINATTR,              /* INPUT      */
             PVOID    pINBUF,               /* INPUT      */
             PBNATTR  pOUTATTR,             /* INPUT      */
             PVOID    pOUTBUF,              /* OUTPUT     */
             PFMTCTOK pFeedBack);           /* OUTPUT     */
```

For the attribute list for PFLATTR, see “Attributes for FLOAT” on page 79. For the attribute list for PBNATTR, see “Attributes for BINARY” on page 77.

Return codes

Message severity	Message Number	Mnemonic	Explanation
0	0	LCF_NO_ERROR	No error.
1	101	LCF_ADL_EXCEPTION_SEV1	**ADL No.** **ADL Exception** 12 Assignment of negative value to unsigned field.
2	102	LCF_ADL_EXCEPTION_SEV2	**ADL No.** **ADL Exception** 11 Fixed-point overflow. 14 Unable to convert. 15 Unable to convert infinity. 18 Assignment of complex to scalar. 21 Fixed-point constraint violation. 22 Fixed-point fit violation.
3	50	LCF_INV_IN_FORM	FORM attribute of input field is invalid.
3	58	LCF_INV_OUT_FIT	FIT attribute of output field is invalid.
3	60	LCF_INV_OUT_LENGTH	LENGTH attribute of output field is invalid.
3	61	LCF_INV_OUT_PRECISION	PRECISION attribute of output field is invalid.
3	62	LCF_INV_OUT_RADIX	RADIX attribute of output field is invalid.
3	63	LCF_INV_OUT_SCALE	SCALE attribute of output field is invalid.
3	64	LCF_INV_OUT_SGNCNV	SGNCNV attribute of output field is invalid.
3	67	LCF_INV_OUT_LENGTH_BYTRVS	LENGTH of input field not multiple of 8 and BYtrvs(TRUE).

FMTFLFL - Float to Float

Purpose

The FMTFLFL function converts an input data field of the ADL FLOAT data type to an output field of the ADL FLOAT data type.

Format

```
void FMTFLFL(PFLATTR  pINATTR,                /* INPUT      */
             PVOID    pINBUF,                 /* INPUT      */
             PFLATTR  pOUTATTR,               /* INPUT      */
             PVOID    pOUTBUF,                /* OUTPUT     */
             PFMTCTOK pFeedBack);             /* OUTPUT     */
```

For the attribute list for PFLATTR, see “Attributes for FLOAT” on page 79.

Return codes

Message severity	Message Number	Mnemonic	Explanation
0	0	LCF_NO_ERROR	No error.
2	102	LCF_ADL_EXCEPTION_SEV2	**ADL No.** **ADL Exception** 5 Floating-point overflow. 13 Floating-point underflow. 14 Unable to convert. 15 Unable to convert infinity. 18 Assignment of complex to scalar. 19 Floating-point fit violation.
3	50	LCF_INV_IN_FORM	FORM attribute of input field is invalid.
3	58	LCF_INV_OUT_FIT	FIT attribute of output field is invalid.
3	60	LCF_INV_OUT_FORM	FORM attribute of input field is invalid.

FMTFLPK - Float to Packed

Purpose

The FMTFLPK function converts an input data field of the ADL FLOAT data type to an output field of the ADL PACKED data type.

Format

```
void FMTFLPK(PFLATTR  pINATTR,                /* INPUT      */
             PVOID    pINBUF,                 /* INPUT      */
             PPKATTR  pOUTATTR,               /* INPUT      */
             PVOID    pOUTBUF,                /* OUTPUT     */
             PFMTCTOK pFeedBack);             /* OUTPUT     */
```

For the attribute list for PFLATTR, see “Attributes for FLOAT” on page 79. For the attribute list for PPKATTR, see “Attributes for PACKED” on page 81.

Return codes

Message severity	Message Number	Mnemonic	Explanation
0	0	LCF_NO_ERROR	No error.
1	101	LCF_ADL_EXCEPTION_SEV1	**ADL No.** **ADL Exception** 12 Assignment of negative value to unsigned field.
2	102	LCF_ADL_EXCEPTION_SEV2	**ADL No.** **ADL Exception** 11 Fixed-point overflow. 14 Unable to convert. 15 Unable to convert infinity. 18 Assignment of complex to scalar. 21 Fixed-point constraint violation. 22 Fixed-point fit violation.
3	50	LCF_INV_IN_FORM	FORM attribute of input field is invalid.
3	58	LCF_INV_OUT_FIT	FIT attribute of output field is invalid.
3	61	LCF_INV_OUT_PRECISION	PRECISION attribute of output field is invalid.
3	63	LCF_INV_OUT_SCALE	SCALE attribute of output field is invalid.

FMTFLZN - Float to Zoned

Purpose

The FMTFLZN function converts an input data field of the ADL FLOAT data type to an output field of the ADL ZONED data type.

Format

```
void FMTFLZN(PFLATTR  pINATTR,              /* INPUT      */
             PVOID    pINBUF,               /* INPUT      */
             PZNATTR  pOUTATTR,             /* INPUT      */
             PVOID    pOUTBUF,              /* OUTPUT     */
             PFMTCTOK pFeedBack);           /* OUTPUT     */
```

For the attribute list for PFLATTR, see “Attributes for FLOAT” on page 79. For the attribute list for PZNATTR, see “Attributes for ZONED” on page 82.

Return codes

Message severity	Message Number	Mnemonic	Explanation
0	0	LCF_NO_ERROR	No error.
1	101	LCF_ADL_EXCEPTION_SEV1	**ADL No.** **ADL Exception** 12 Assignment of negative value to unsigned field.
2	102	LCF_ADL_EXCEPTION_SEV2	**ADL No.** **ADL Exception** 11 Fixed-point overflow. 14 Unable to convert. 15 Unable to convert infinity. 18 Assignment of complex to scalar. 21 Fixed-point constraint violation. 22 Fixed-point fit violation.
3	103	LCF_ADL_EXCEPTION_SEV3	**ADL No.** **ADL Exception** 2 CCSID not supported.
3	50	LCF_INV_IN_FORM	FORM attribute of input field is invalid.
3	58	LCF_INV_OUT_FIT	FIT attribute of output field is invalid.
3	61	LCF_INV_OUT_PRECISION	PRECISION attribute of output field is invalid.
3	63	LCF_INV_OUT_SCALE	SCALE attribute of output field is invalid.
3	65	LCF_INV_OUT_SGNLOC	SGNLOC attribute of output field is invalid.
3	66	LCF_INV_OUT_ZONENC	ZONENC attribute of output field is invalid.

FMTPKBN - Packed to Binary

Purpose

The FMTPKBN function converts an input data field of the ADL PACKED data type to an output field of the ADL BINARY data type.

Format

```
void FMTPKBN(PPKATTR  pINATTR,               /* INPUT    */
             PVOID    pINBUF,                /* INPUT    */
             PBNATTR  pOUTATTR,              /* INPUT    */
             PVOID    pOUTBUF,               /* OUTPUT   */
             PFMTCTOK pFeedBack);            /* OUTPUT   */
```

For the attribute list for PPKATTR, see “Attributes for PACKED” on page 81. For the attribute list for PBNATTR, see “Attributes for BINARY” on page 77.

Return codes

Message severity	Message Number	Mnemonic	Explanation
0	0	LCF_NO_ERROR	No error.
1	101	LCF_ADL_EXCEPTION_SEV1	**ADL No.** **ADL Exception** 12 Assignment of negative value to unsigned field.
2	102	LCF_ADL_EXCEPTION_SEV2	**ADL No.** **ADL Exception** 11 Fixed-point overflow. 18 Assignment of complex to scalar. 21 Fixed-point constraint violation. 22 Fixed-point fit violation.
3	103	LCF_ADL_EXCEPTION_SEV3	**ADL No.** **ADL Exception** 35 Invalid digit in PACKED input field (digit>9). 36 Invalid sign in PACKED input field.
3	52	LCF_INV_IN_PRECISION	PRECISION attribute of input field is invalid.
3	54	LCF_INV_IN_SCALE	SCALE attribute of input field is invalid.
3	58	LCF_INV_OUT_FIT	FIT attribute of output field is invalid.
3	60	LCF_INV_OUT_LENGTH	LENGTH attribute of output field is invalid.
3	61	LCF_INV_OUT_PRECISION	PRECISION attribute of output field is invalid.
3	62	LCF_INV_OUT_RADIX	RADIX attribute of output field is invalid.
3	63	LCF_INV_OUT_SCALE	SCALE attribute of output field is invalid.
3	64	LCF_INV_OUT_SGNCNV	SGNCNV attribute of output field is invalid.
3	67	LCF_INV_OUT_LENGTH_BYTRVS	LENGTH of input field not multiple of 8 and BYTRVS(TRUE).

FMTPKFL - Packed to Float

Purpose

The FMTPKFL function converts an input data field of the ADL PACKED data type to an output field of the ADL FLOAT data type.

Format

```
void FMTPKFL(PPKATTR  pINATTR,              /* INPUT    */
             PVOID    pINBUF,               /* INPUT    */
             PFLATTR  pOUTATTR,             /* INPUT    */
             PVOID    pOUTBUF,              /* OUTPUT   */
             PFMTCTOK pFeedBack);           /* OUTPUT   */
```

For the attribute list for PPKATTR, see “Attributes for PACKED” on page 81. For the attribute list for PFLATTR, see “Attributes for FLOAT” on page 79.

Return codes

Message severity	Message Number	Mnemonic	Explanation
0	0	LCF_NO_ERROR	No error.
2	102	LCF_ADL_EXCEPTION_SEV2	**ADL No.** **ADL Exception** 5 Floating-point overflow. 13 Floating-point underflow. 18 Assignment of complex to scalar. 19 Floating-point fit violation.
3	103	LCF_ADL_EXCEPTION_SEV3	**ADL No.** **ADL Exception** 35 Invalid digit in PACKED input field (digit>9). 36 Invalid sign in PACKED input field.
3	52	LCF_INV_IN_PRECISION	PRECISION attribute of input field is invalid.
3	54	LCF_INV_IN_SCALE	SCALE attribute of input field is invalid.
3	58	LCF_INV_OUT_FIT	FIT attribute of output field is invalid.
3	59	LCF_INV_OUT_FORM	FORM attribute of input field is invalid.

FMTPKPK - Packed to Packed

Purpose

The FMTPKPK function converts an input data field of the ADL PACKED data type to an output field of the ADL PACKED data type.

Format

```
void FMTPKPK(PPKATTR  pINATTR,               /* INPUT    */
             PVOID    pINBUF,                /* INPUT    */
             PPKATTR  pOUTATTR,              /* INPUT    */
             PVOID    pOUTBUF,               /* OUTPUT   */
             PFMTCTOK pFeedBack);            /* OUTPUT   */
```

For the definition of the data type PPKATTR see “Attributes for PACKED” on page 81.

Return codes

Message severity	Message Number	Mnemonic	Explanation
0	0	LCF_NO_ERROR	No error.
1	101	LCF_ADL_EXCEPTION_SEV1	**ADL No.** **ADL Exception** 12 Assignment of negative value to unsigned field.
2	102	LCF_ADL_EXCEPTION_SEV2	**ADL No.** **ADL Exception** 11 Fixed-point overflow. 18 Assignment of complex to scalar. 21 Fixed-point constraint violation. 22 Fixed-point fit violation.
3	103	LCF_ADL_EXCEPTION_SEV3	**ADL No.** **ADL Exception** 35 Invalid digit in PACKED input field (digit>9). 36 Invalid sign in PACKED input field.
3	52	LCF_INV_IN_PRECISION	PRECISION attribute of input field is invalid.
3	54	LCF_INV_IN_SCALE	SCALE attribute of input field is invalid.
3	58	LCF_INV_OUT_FIT	FIT attribute of output field is invalid.
3	61	LCF_INV_OUT_PRECISION	PRECISION attribute of output field is invalid.
3	63	LCF_INV_OUT_SCALE	SCALE attribute of output field is invalid.

FMTPKZN - Packed to Zoned

Purpose

The FMTPKZN function converts an input data field of the ADL PACKED data type to an output field of the ADL ZONED data type.

Format

```
void FMTPKZN(PPKATTR  pINATTR,               /* INPUT      */
             PVOID    pINBUF,                /* INPUT      */
             PZNATTR  pOUTATTR,              /* INPUT      */
             PVOID    pOUTBUF,               /* OUTPUT     */
             PFMTCTOK pFeedBack);            /* OUTPUT     */
```

For the attribute list for PPKATTR, see “Attributes for PACKED” on page 81. For the attribute list for PZNATTR, see “Attributes for ZONED” on page 82.

Return codes

Message severity	Message Number	Mnemonic	Explanation
0	0	LCF_NO_ERROR	No error.
1	101	LCF_ADL_EXCEPTION_SEV1	**ADL No.** **ADL Exception** 12 Assignment of negative value to unsigned field.
2	102	LCF_ADL_EXCEPTION_SEV2	**ADL No.** **ADL Exception** 11 Fixed-point overflow. 18 Assignment of complex to scalar. 21 Fixed-point constraint violation. 22 Fixed-point fit violation.
3	103	LCF_ADL_EXCEPTION_SEV3	**ADL No.** **ADL Exception** 2 CCSID not supported. 35 Invalid digit in PACKED input field (digit>9). 36 Invalid sign in PACKED input field.
3	52	LCF_INV_IN_PRECISION	PRECISION attribute of input field is invalid.
3	54	LCF_INV_IN_SCALE	SCALE attribute of input field is invalid.
3	58	LCF_INV_OUT_FIT	FIT attribute of output field is invalid.
3	52	LCF_INV_OUT_PRECISION	PRECISION attribute of output field is invalid.
3	63	LCF_INV_OUT_SCALE	SCALE attribute of output field is invalid.
3	65	LCF_INV_OUT_SGNLOC	SGNLOC attribute of output field is invalid.
3	66	LCF_INV_OUT_ZONENC	ZONENC attribute of output field is invalid.

FMTZNBN - Zoned to Binary

Purpose

The FMTZNBN function converts an input data field of the ADL ZONED data type to an output field of the ADL BINARY data type.

Format

```
void FMTZNBN(PZNATTR  pINATTR,               /* INPUT      */
             PVOID    pINBUF,                /* INPUT      */
             PBNATTR  pOUTATTR,              /* INPUT      */
             PVOID    pOUTBUF,               /* OUTPUT     */
             PFMTCTOK pFeedBack);            /* OUTPUT     */
```

For the attribute list for PZNATTR, see “Attributes for ZONED” on page 82. For the attribute list for PBNATTR, see “Attributes for BINARY” on page 77.

Return codes

Message severity	Message Number	Mnemonic	Explanation
0	0	LCF_NO_ERROR	No error.
1	101	LCF_ADL_EXCEPTION_SEV1	**ADL No.** **ADL Exception** 12 Assignment of negative value to unsigned field.
2	102	LCF_ADL_EXCEPTION_SEV2	**ADL No.** **ADL Exception** 11 Fixed-point overflow. 18 Assignment of complex to scalar. 21 Fixed-point constraint violation. 22 Fixed-point fit violation.
3	103	LCF_ADL_EXCEPTION_SEV3	**ADL No.** **ADL Exception** 2 CCSID not supported. 37 Invalid digit in ZONED input field (digit>9). 38 Invalid sign in ZONED input field. 39 Invalid zone nibble in ZONED input field.
3	52	LCF_INV_IN_PRECISION	PRECISION attribute of input field is invalid.
3	54	LCF_INV_IN_SCALE	SCALE attribute of input field is invalid.
3	55	LCF_INV_IN_SGNLOC	SGNLOC attribute of input field is invalid.
3	56	LCF_INV_IN_ZONENC	ZONENC attribute of input field is invalid.
3	58	LCF_INV_OUT_FIT	FIT attribute of output field is invalid.
3	60	LCF_INV_OUT_LENGTH	LENGTH attribute of output field is invalid.
3	61	LCF_INV_OUT_PRECISION	PRECISION attribute of output field is invalid.
3	62	LCF_INV_OUT_RADIX	RADIX attribute of output field is invalid.

FMTZNBN

Message severity	Message Number	Mnemonic	Explanation
3	63	LCF_INV_OUT_SCALE	SCALE attribute of output field is invalid.
3	64	LCF_INV_OUT_SGNCNV	SGNCNV attribute of output field is invalid.
3	67	LCF_INV_OUT_LENGTH_BYTRVS	LENGTH of input field not multiple of 8 and BYTRVS(TRUE).

FMTZNFL - Zoned to Float

Purpose

The FMTZNFL function converts an input data field of the ADL ZONED data type to an output field of the ADL FLOAT data type.

Format

```
void FMTZNFL(PZNATTR  pINATTR,               /* INPUT      */
             PVOID    pINBUF,                /* INPUT      */
             PFLATTR  pOUTATTR,              /* INPUT      */
             PVOID    pOUTBUF,               /* OUTPUT     */
             PFMTCTOK pFeedBack);            /* OUTPUT     */
```

For the attribute list for PZNATTR, see “Attributes for ZONED” on page 82. For the attribute list for PFLATTR, see “Attributes for FLOAT” on page 79.

Return codes

Message severity	Message Number	Mnemonic	Explanation
0	0	LCF_NO_ERROR	No error.
2	102	LCF_ADL_EXCEPTION_SEV2	**ADL No.** **ADL Exception** 5 Floating-point overflow. 13 Floating-point underflow. 18 Assignment of complex to scalar. 19 Floating-point fit violation.
3	103	LCF_ADL_EXCEPTION_SEV3	**ADL No.** **ADL Exception** 2 CCSID not supported. 37 Invalid digit in ZONED input field (digit>9). 38 Invalid sign in ZONED input field. 39 Invalid zone nibble in ZONED input field.
3	52	LCF_INV_IN_PRECISION	PRECISION attribute of input field is invalid.
3	54	LCF_INV_IN_SCALE	SCALE attribute of input field is invalid.
3	55	LCF_INV_IN_SGNLOC	SGNLOC attribute of input field is invalid.
3	56	LCF_INV_IN_ZONENC	ZONENC attribute of input field is invalid.
3	58	LCF_INV_OUT_FIT	FIT attribute of output field is invalid.
3	59	LCF_INV_OUT_FORM	FORM attribute of input field is invalid.

FMTZNPK - Zoned to Packed

Purpose

The FMTZNPK function converts an input data field of the ADL ZONED data type to an output field of the ADL PACKED data type.

Format

```
void FMTZNPK(PZNATTR  pINATTR,                /* INPUT     */
             PVOID    pINBUF,                 /* INPUT     */
             PPKATTR  pOUTATTR,               /* INPUT     */
             PVOID    pOUTBUF,                /* OUTPUT    */
             PFMTCTOK pFeedBack);             /* OUTPUT    */
```

For the attribute list for PZNATTR, see “Attributes for ZONED” on page 82. For the attribute list for PPKATTR, see “Attributes for PACKED” on page 81.

Return codes

Message severity	Message Number	Mnemonic	Explanation
0	0	LCF_NO_ERROR	No error.
1	101	LCF_ADL_EXCEPTION_SEV1	**ADL No.** **ADL Exception** 12 Assignment of negative value to unsigned field.
2	102	LCF_ADL_EXCEPTION_SEV2	**ADL No.** **ADL Exception** 11 Fixed-point overflow. 18 Assignment of complex to scalar. 21 Fixed-point constraint violation. 22 Fixed-point fit violation.
3	103	LCF_ADL_EXCEPTION_SEV3	**ADL No.** **ADL Exception** 2 CCSID not supported. 37 Invalid digit in ZONED input field (digit>9). 38 Invalid sign in ZONED input field. 39 Invalid zone nibble in ZONED input field.
3	52	LCF_INV_IN_PRECISION	PRECISION attribute of input field is invalid.
3	54	LCF_INV_IN_SCALE	SCALE attribute of input field is invalid.
3	55	LCF_INV_IN_SGNLOC	SGNLOC attribute of input field is invalid.
3	56	LCF_INV_IN_ZONENC	ZONENC attribute of input field is invalid.
3	58	LCF_INV_OUT_FIT	FIT attribute of output field is invalid.
3	61	LCF_INV_OUT_PRECISION	PRECISION attribute of output field is invalid.
3	63	LCF_INV_OUT_SCALE	SCALE attribute of output field is invalid.

FMTZNZN - Zoned to Zoned

Purpose

The FMTZNZN function converts an input data field of the ADL ZONED data type to an output field of the ADL ZONED data type.

Format

```
void FMTZNZN(PZNATTR  pINATTR,                /* INPUT      */
             PVOID    pINBUF,                 /* INPUT      */
             PZNATTR  pOUTATTR,               /* INPUT      */
             PVOID    pOUTBUF,                /* OUTPUT     */
             PFMTCTOK pFeedBack);             /* OUTPUT     */
```

For the definition of the data type PZNATTR see “Attributes for ZONED” on page 82.

Return codes

Message severity	Message Number	Mnemonic	Explanation
0	0	LCF_NO_ERROR	No error.
1	101	LCF_ADL_EXCEPTION_SEV1	**ADL No.** **ADL Exception** 12 Assignment of negative value to unsigned field.
2	102	LCF_ADL_EXCEPTION_SEV2	**ADL No.** **ADL Exception** 11 Fixed-point overflow. 18 Assignment of complex to scalar. 21 Fixed-point constraint violation. 22 Fixed-point fit violation.
3	103	LCF_ADL_EXCEPTION_SEV3	**ADL No.** **ADL Exception** 2 CCSID not supported. 37 Invalid digit in ZONED input field (digit>9). 38 Invalid sign in ZONED input field. 39 Invalid zone nibble in ZONED input field.
3	52	LCF_INV_IN_PRECISION	PRECISION attribute of input field is invalid.
3	54	LCF_INV_IN_SCALE	SCALE attribute of input field is invalid.
3	55	LCF_INV_IN_SGNLOC	SGNLOC attribute of input field is invalid.
3	56	LCF_INV_IN_ZONENC	ZONENC attribute of input field is invalid.
3	58	LCF_INV_OUT_FIT	FIT attribute of output field is invalid.
3	61	LCF_INV_OUT_PRECISION	PRECISION attribute of output field is invalid.
3	63	LCF_INV_OUT_SCALE	SCALE attribute of output field is invalid.
3	65	LCF_INV_OUT_SGNLOC	SGNLOC attribute of output field is invalid.
3	66	LCF_INV_OUT_ZONENC	ZONENC attribute of output field is invalid.

Chapter 7. The User Exit — Calling Your Own Programs

Purpose

The DD&C user–exit allows you to call your own programs while executing a plan statement.

The format of the CALL statement required to include the user–exit function in ADL source text is described in *SMARTdata UTILITIES A Data Language Reference for Data Description and Conversion*.

The user-provided program is called at conversion plan execution time.

Format

```
void MyFunction(LONG     lParamCount,    /* INPUT      */
                VOID     *rgpParam[ ],   /* INPUT/OUTPUT */
                PFMTCTOK pFeedBack);     /* INPUT/OUTPUT */
```

Parameters

MyFunction
The function name which should be the entry point to the program name that was specified on the ADL CALL statement.

lParamCount
An input variable that contains the number of parameters specified in the ADL CALL statement.

rgpParam
An array of pointers to the parameters specified in the ADL call statement. The pointers are set to the locations of the parameters in the order they are specified in the ADL CALL statement.

pFeedBack
The start address of an area in which the user-exit function writes error information. This condition token can contain a message severity and message number indicating the most severe error encountered during the processing of the function. See “The Condition Token” on page 20 for the layout and data type description of the condition token.

When calling the user-exit function, DD&C passes the data area **pI_S_INFO**, initialized by the caller of FMTCPXC, to the user exit as the pFeedback buffer.

Return Codes

Execution of the conversion plan stops when the severity code, returned by the user-provided function in the pFeedback buffer, is greater than 1 (Warning).

Chapter 8. Data Conversion Exceptions and Errors

This chapter discusses methods of finding and correcting errors that can occur during data conversion. It includes:

- Finding and correcting errors
- Using the condition token to extract error messages
- Using the listing file to identify errors
- Using the consistence token to control ADLDCLSPCs
- Using the generate function to check ADL source files

It also lists:

- ADL exceptions
- CDRA return codes
- CDRA return codes mapped to DD&C return codes.
- Parse and Generate function messages.

Finding and Correcting Errors in Your Programs

The following describes how you can interpret and correct errors detected by DD&C in your programs. The methods you can use to do this include using:

- The condition token
- The listing file
- The consistency token
- The generate function.

Table 8 summarizes the diagnostic information that is returned by the DD&C API functions and tells you where to find further information.

Table 8. DD&C Diagnostic Information

API function	Information returned	Where to look
FMTPRS	Listing file messages. Message severity and number.	"Parse Function Messages" on page 119. Return codes on page 32.
FMTGEN	Listing file messages. Message severity and number.	"Generate Function Messages" on page 141. Return codes on page 36.
FMTCRCP, FMTCPXI, FMTCPXC	ADL exceptions. Message severity and number.	"ADL Exceptions" on page 112. "Return codes" of Chapter 4 and Chapter 5.
FMTCPXT	Message severity and number.	Return codes on page 57.
Alphanumeric conversion routines	Status and reason codes.	"Return codes" of alphanumeric conversion routines in Chapter 6. "CDRA Return Codes" on page 115.
Numeric conversion routines	ADL exceptions. Message severity and number.	"ADL Exceptions" on page 112. "Numeric Conversion Routines" on page 76.
Trace facility	Error messages.	"Trace Function Messages" on page 225

Extracting Error Messages From the Condition Token

The condition token is returned by every DD&C API function. The information returned in the condition token (the structure of which is described in “The Condition Token” on page 20) includes:

- The message severity
- The message number
- For numeric data-type conversion routines and FMTCPXI, the ADL exception identifier
- For FMTCRCP and FMTCPXC, a pointer to the ADL communications area.

Your program should extract and test the appropriate fields of the condition token after every API function call. Each API function description includes a table listing all possible values of message severity and message number (or status and reason codes for the CDRA functions). Use the explanation column of these tables for advice on resolving the problem.

For the numeric data-type conversion routines and the FMTCPXI function, ADL exceptions are returned in the **pFeedBack->pI_S_Info.ulAdlExId** field of the condition token. Otherwise, ADL exceptions are returned in the **pFeedBack->pI_S_Info.pAdlCommArea->lExId** field of the ADL communications area (FMTADLCA). “ADL Exceptions” on page 112 contains a complete list of the ADL exceptions that can occur, together with an explanation of the possible cause.

Using the Listing File to Identify Errors

Use the Parse (FMTPRS) and Generate (FMTGEN) functions to create a listing file.

For the Parse function, you can specify whether the listing file includes both the ADL source code being parsed and any error messages detected during parsing, or the error messages only.

You can specify the severity of messages that are included in the listing on the FLAG parameter.

Specify **I** to include all error messages.
Specify **W** to include warning, error and severe messages.
Specify **E** to include error and severe messages.
Specify **S** to include only severe messages.

Error message listings begin on 119.

Figure 13 shows an extract of a listing file with a number of ADL errors detected during parsing. This listing file was produced using the options LIST and FLAG(I).

```
(0:0): informational FMT1000: Data Description and Conversion for OS/2 Version 1.10
(0:0): informational FMT1001: ADL Parser (c) Copyright IBM Corp. 1994.
(0:0): informational FMT1002: All rights reserved.
<0001> DECLARE BEGIN;
<0002>    COBOLREC: SEQUENCE BEGIN;
<0003>       INITIALS: CHAR LENGTH(4) BYTRVS(TRUE);
<0004>       NUMBER: PACKED PRECISION(5);
<0005>    END;
<0006> END;
sample.adl(3:32): error FMT1453: This attribute is not allowed with a CHAR object.
<0007> DECLARE BEGIN;
<0008>    A: CONSTANT B;
<0009>    B: CONSTANT A;
<0010>    CREC: SEQUENCE BEGIN;
<0011>       INITIALS: CHARSFX MAXLEN(A) CCSID(850);
<0012>       NUMBER: BINARY PRECISION(15) BYTRVS(TRUE);
<0013>    END;
<0014> END;
sample.adl(9:4): error FMT1418: Constant declaration loop for "B".
```

Figure 13. Sample Parse Listing File

Use the Generate function, to create either:

A symbolic representation of the ADL declare space, ADLDCLSPC,
A listing of error messages similar to that of the Parse function,
A combination of both.

An example of the symbolic representation of ADLDCLSPC is shown in Figure 17 on page 155. This is generally used by IBM service personnel only. Generate function messages begin on 141

Using the Consistency Token to Control ADLDCLSPCs

The consistency token is calculated by the Parse function (FMTPRS) and reflects the physical composition of the ADLDCLSPC. If the ADL source code passed to the Parse function is structurally changed, the composition of the ADLDCLSPC changes and so does the value of the consistency token returned.

By comparing the consistency token returned from successive parsings of the same file, therefore, you can determine whether the source file has been modified since the last time it was parsed. If you need to find out how the source file has been modified, you can use the Generate function to create ADL source code from the ADLDCLSPC and compare it to the original source.

Using the Generate Function to Create ADL Source Files

You can use the Generate function to test that the composition of an ADLDCLSPC is correct. The Generate function produces ADL source code from an ADLDCLSPC. If the ADLDCLSPC is correct, this source code should be equivalent to that passed as an input parameter to the Parse function. One important difference, however, is that the ADL source code produced by the Generate function includes all default ADL attributes, even if these were not specified in the ADL source code passed to the Parse function. Therefore, you can use the Generate function to check the ADL defaults that are implemented by DD&C. For an example of this, see Figure 14 on page 146 and Figure 18 on page 158, which show sample ADL source files input to the Parse function and output by the Generate function, respectively.

ADL Exceptions

A unique value is assigned to each ADL exception.

Message severity	Message number	Mnemonic	Explanation
3	1	ADL_CONV_NOT_SUPPORTED	Conversion not supported. The attempted conversion is not supported in the conversion matrix shown in *SMARTdata UTILITIES A Data Language Reference for Data Description and Conversion*.
3	2	ADL_CCSID_NOT_SUPPORTED	CCSID not supported. The source or target CCSID is not supported.
3	3	ADL_INVALID_CCSID_PAIR	Invalid CCSID pair. Conversion between the source CCSID and the target CCSID is not supported.
3	4	ADL_UNDEFINED_CCSID	Undefined CCSID. The source or target CCSID is not defined by CDRA.
2	5	ADL_FLOAT_OVERFLOW	Floating-point overflow.
3	6	ADL_CASE_FAILURE	Target CASE failure. The selected target <WHEN statement> evaluates to FALSE or one or more target <WHEN statement>s preceding the selected <WHEN statement> evaluates to TRUE.
3	8	ADL_NONCONFORM_ARRAYS	Nonconformable arrays. The target array does not have the same number of dimensions as the source array, or the dimension sizes of a dimension do not match.
3	9	ADL_ENUM_MISMATCH	ENUMERATION mismatch. The source enumeration identifier does not match an enumeration identifier of the target.
3	10	ADL_INVALID_ENUM_VALUE	Invalid ENUMERATION value. The value of the source number does not match a value associated with the enumeration identifier of the target.
2	11	ADL_FIX_OVERFLOW	Fixed-point overflow. The converted source number is too large to be represented within the target field.
1	12	ADL_NEG_TO_UNSIGNED	Assignment of negative value to unsigned field. The source is a signed negative number and the target field is unsigned.
2	14	ADL_NOT_A_NUMBER	Unable to convert. The source value is not a “Not a Number” (NaN).
2	15	ADL_INFINITY	Unable to convert infinity. The source value represents infinity and cannot be converted.
3	16	ADL_INPUT_AREA_TOO_SHORT	Input area too short. The length specified on an input parameter of the plan is less than the declared length of the data.

Message severity	Message number	Mnemonic	Explanation
3	17	ADL_OUTPUT_AREA_TOO_SHORT	Output area too short. The maximum length specified on an output parameter of the plan is less than the declared length of the data.
2	18	ADL_COMPLEX_TO_SCALAR	Assignment of complex to scalar. The source contains a complex number but the target field is not a complex number.
2	19	ADL_FLOAT_FIT_VIOLATION	Floating-point fit violation. The converted number results in the loss of low-order digits in a target floating-point field with FIT(EXACT) specified.
3	20	ADL_CASE_REJECTED	CASE rejected. All <WHEN statement>s evaluate to FALSE and an <OTHERWISE statement> does not exist.
2	21	ADL_FIX_CONSTRAINT_VIOLATION	Fixed-point constraint violation. The number of significant digits required in the stored value exceeds the number of digits specified by the PRECISION attribute of the target with CONSTRAINED(TRUE) specified.
2	22	ADL_FIX_FIT_VIOLATION	Fixed-point fit violation. The converted number results in the loss of low-order digits in a target fixed-point field with FIT(EXACT) specified.
2	23	ADL_SEQ_ELEMENT_NOT_FOUND	Sequence element not found. A SEQUENCE that is the target of an assignment contains an element for which an element of the source SEQUENCE with a matching identifier cannot be found.
3	24	ADL_TARGET_CASE_MISMATCH	Target CASE mismatch. A source <WHEN statement> evaluates to TRUE, but no target <WHEN statement> with the same statement <identifier> or <positional identifier> can be found.
3	25	ADL_NEG_ARRAY_SIZE	Negative array dimension size. The size of a source array dimension was specified by a referenced field or was calculated. This value is negative.
3	26	ADL_INVALID_ARRAY_SIZE	Invalid array dimension size. The size of a source array dimension was specified by a referenced field or was calculated. This value is greater than the value of the <DMNMAX attribute>.
3	27	ADL_INV_IN_LENGTH	Incorrect LENGTH value of ASIS, BIT, or BITPRE field. The LENGTH value is greater than MAXLEN or the remaining buffer size.
3	28	ADL_INV_IN_HIGH_LOW	Incorrect LENGTH or HIGH LOW value of CHAR or CHARPRE field. The length of the field is greater than MAXLEN or the remaining buffer size.
3	29	ADL_INV_OUT_HIGH_LOW	Incorrect LENGTH or HIGH LOW value of CHAR or CHARPRE field. The length of the field is greater than the MAXLEN size.
3	30	ADL_IN_NO_SFX	Input CHARSFX field contains no suffix.

Message severity	Message number	Mnemonic	Explanation
3	31	ADL_OUT_MULT_SFX	Output CHARSFX field might contain one or more characters matching the suffix.
3	32	ADL_IN_ORPHAN_BYTE	Input is DBCS and orphan byte was found.
3	33	ADL_OUT_ORPHAN_BYTE	Output is DBCS and output length is odd.
3	34	ADL_INV_IN_STRING	Invalid input character field.
3	35	ADL_INV_PK_DIGIT	Invalid digit in PACKED input field (digit>9).
3	36	ADL_INV_PK_SIGN	Invalid sign in PACKED input field.
3	37	ADL_INV_ZN_DIGIT	Invalid digit in ZONED input field (digit>9).
3	38	ADL_INV_ZN_SIGN	Invalid sign in ZONED.
3	39	ADL_INV_ZN_ZONE	Invalid zone nibble in ZONED input field.

CDRA Return Codes

Like the other DD&C API functions, the CDRA data-type conversion routines return the processing status in the condition token. Instead of the message severity and message number, however a status code and a reason code are returned. The status codes that are returned have the following meaning:

Status (Hex)	**Reason**
X'0000'	The function completed successfully.
X'0001'	An element or value was not found while the function was running.
X'0002'	A CCSID, code page, or character set value of 0 was encountered.
X'0003'	The CCSID encountered is a special-purpose CCSID in the range 65280 (X'FF00') to 65535 (X'FFFF'). This value is also used to indicate a code page value of 65535 (X'FFFF').
X'0004'	An overflow situation was encountered. See “Data Overflow” on page 25.
X'0005'	A syntax error was detected in one or more parameters.
X'0006'	The function encountered a condition that prevents it from proceeding. For example, the CCSID resource table was not found, or insufficient storage was available to be able to copy or load resources.
X'0007'	One or more CDRA resources required by the function are damaged and cannot be used.
X'0008'	A parameter's value is outside the specified range.
X'0009' to X'00FF'	Reserved for future allocation by CDRA.
X'0100'	Indicates that a graphic character substitution occurred, resulting in a loss of information.
X'0101' to X'07FF'	Not used in DD&C.
X'0800'	OS/2 environment-specific return codes.
X'0801' to X'17FF'	Not used in DD&C.
X'1800' to X'7FFF'	Function-specific and environment-specific, not used by DD&C.
X'8000' to X'FFFF'	Not used in DD&C.

The CDRA return codes actually returned are listed separately in the “Return codes” section of each API function description. Table 9 shows how Conversion Plan Executor error messages are mapped to their CDRA-defined equivalents for the CDRMSCI function.

Table 9 (Page 1 of 3). How CDRA messages are mapped to Conversion Plan Executor messages

CDRA			Conversion Plan Executor
Status code	**Reason code**	**Explanation**	**Mnemonic**
X'0000'	X'0000'	The function completed successfully.	CPX_NO_ERROR
X'0001'	X'0001'	The requested conversion is not supported (there is no entry in the GCCST) for the specified combination of CCSID1, ST1, CCSID2, ST2, and GCCASN.	CPX_ADL_EXCEPTION_SEV3 (ADL_INVALID_CCSID_PAIR)
X'0001'	X'0005'	The requested conversion algorithm specified by GCCASN does not support the specified (CCSID1, ST1) to (CCSID2, ST2) combination.	CPX_INTERNAL_ERR
X'0001'	X'0006'	GCCASN value is 0, but an "installation default" was not found in the GCCST for the pair (CCSID1, ST1) to (CCSID2, ST2).	CPX_ADL_EXCEPTION_SEV3 (ADL_INVALID_CCSID_PAIR)
X'0002'	X'0001'	CCSID1 is zero, meaning that the real CCSID pertaining to the caller's data must be determined by the caller from the next higher level in some hierarchy. The invoking program must resolve the default before invoking this function.	CPX_INTERNAL_ERR
X'0002'	X'0002'	CCSID2 is zero, which is reserved to indicate defaulting to a higher level in a hierarchy. The invoking program must resolve the default before invoking this function.	CPX_INTERNAL_ERR
X'0003'	X'0001'	CCSID1 is one of the special-purpose CCSID values in the range 65280 to 65535.	CPX_ADL_EXCEPTION_SEV3 (ADL_CCSID_NOT_SUPPORTED)
X'0003'	X'0002'	CCSID2 is one of the special-purpose CCSID values in the range 65280 to 65535.	CPX_ADL_EXCEPTION_SEV3 (ADL_CCSID_NOT_SUPPORTED)
X'0004'	X'0001'	The supplied output buffer was too small for the output data. A properly truncated and terminated converted string that fits within the allocated maximum is returned in the area starting at S2 with its byte-length in L3. Value in L4 is set to the first byte of the code point, representing the next character to be converted in the input string S1.	CPX_INTERNAL_ERR
X'0004'	X'0002'	The encoding scheme of CCSID1 is X'1301' (Mixed Host SB/DB encoding). The length value in L2 allocated for area S2 was too small for the output data. A properly truncated and terminated converted string that fits within the allocated maximum is returned in the area starting at S2 with its byte-length in L3. Value in L4 is set to the first byte of a double byte character (between SO and SI brackets) that would have been converted next in the input buffer.	CPX_INTERNAL_ERR
X'0005'	X'0001'	A pure DBCS CCSID1 was specified, but either ST1 is equal to 0 and L1 is odd, or ST1 is equal to 1 and an orphan byte was detected.	CPX_ADL_EXCEPTION_SEV3 (ADL_IN_ORPHAN_BYTE)

Table 9 (Page 2 of 3). How CDRA messages are mapped to Conversion Plan Executor messages

CDRA			Conversion Plan Executor
Status code	**Reason code**	**Explanation**	**Mnemonic**
X'0005'	X'0004'	The ES of CCSID1 was X'1301' and an incorrectly formed string, (an odd number of bytes between SO, SI brackets) was encountered.	CPX_ADL_EXCEPTION_SEV3 (ADL_INV_IN_STRING)
X'0005'	X'0005'	ST1 is equal to 1 but a null-termination character was not found in the input buffer.	CPX_ADL_EXCEPTION_SEV3 (ADL_IN_NO_SFX)
X'0005'	X'0006'	ST2 is equal to 1. However, the output string contains one or more characters matching the null-termination character, resulting from using the selected conversion tables.	CPX_ADL_EXCEPTION_SEV3 (ADL_OUT_MULT_SFX)
X'0005'	X'0007'	ST2 is equal to 2, but the required space character is not defined in the CDRA resources, or the CCSID resource definition could not be found.	CPX_ADL_EXCEPTION_SEV3 (ADL_INVALID_CCSID_PAIR)
X'0005'	X'0008'	A pure double-byte CCSID2 with ST2 equal to 1 was specified, and an odd value was specified for length L2 of the output buffer. The conversion function returns only an even number of bytes (maximum L2-1 bytes), including the null-termination character in ST2. The contents of the remaining locations in the output buffer are unpredictable.	CPX_ADL_EXCEPTION_SEV3 (ADL_OUT_ORPHAN_BYTE)
X'0005'	X'0009'	A pure double-byte CCSID2 with ST2 equal to 2 was specified (a space-padded string), and an odd value was specified for length L2 of the output buffer. The conversion function returns L2-1 bytes, including the space-padded characters in ST2. The contents of the remaining locations in the output buffer are unpredictable.	CPX_ADL_EXCEPTION_SEV3 (ADL_OUT_ORPHAN_BYTE)
X'0005'	X'000C'	The ES of CCSID1 was X'1301', but a trailing SI bracket is missing.	CPX_ADL_EXCEPTION_SEV3 (ADL_INV_IN_STRING)
X'0005'	X'000D'	The ES of CCSID1 was X'1301' and a trailing SI code point was encountered without first encountering a corresponding SO code point. The number of intervening code points may have been odd or even—the code points are treated as single-byte code points because the leading SO was missing)	CPX_ADL_EXCEPTION_SEV3 (ADL_INV_IN_STRING)
X'0006'	X'0001'	The GCCST was not found.	CPX_CDRA_RESOURCE_ERROR
X'0006'	X'0002'	A CDRA resource is currently unavailable.	CPX_RESOURCE_LIM
X'0006'	X'0003'	The conversion method identified in the GCCST for the specified selection is currently unavailable.	CPX_RESOURCE_LIM
X'0006'	X'0004'	A conversion table identified in the GCCST for the specified selection is not found.	CPX_CDRA_RESOURCE_ERROR
X'0006'	X'0006'	The token's structure is incorrect.	CPX_INTERNAL_ERR
X'0006'	X'0007'	Unable to generate the token as requested.	CPX_NO_MEMORY

Table 9 (Page 3 of 3). How CDRA messages are mapped to Conversion Plan Executor messages

CDRA			Conversion Plan Executor
Status code	**Reason code**	**Explanation**	**Mnemonic**
X'0007'	X'0001'	The structure of the GCCST resource accessed by the function is incorrect.	CPX_CDRA_RESOURCE_ERROR
X'0007'	X'0002'	The structure of the system GCCT resource accessed by the function is incorrect.	CPX_CDRA_RESOURCE_ERROR
X'0007'	X'0003'	The table type of GCCT does not match the method selected from the GCCST.	CPX_CDRA_RESOURCE_ERROR
X'0007'	X'0004'	There is no ES element definition in the CCSID resource for lCCSID1.	CPX_CDRA_RESOURCE_ERROR
X'0008'	X'0001'	The value of CCSID1 is outside the permitted range.	CPX_ADL_EXCEPTION_SEV3 (ADL_UNDEFINED_CCSID)
X'0008'	X'0002'	The value of CCSID2 is outside the permitted range.	CPX_ADL_EXCEPTION_SEV3 (ADL_UNDEFINED_CCSID)
X'0008'	X'0003'	The value of ST1 is outside the permitted range.	CPX_INTERNAL_ERR
X'0008'	X'0004'	The value of ST2 is outside the permitted range.	CPX_INTERNAL_ERR
X'0008'	X'0005'	The value of IL1 is outside the permitted range.	CPX_INTERNAL_ERR
X'0008'	X'0006'	The value of IL2 is outside the permitted range.	CPX_INTERNAL_ERR
X'0008'	X'0007'	The value of GCCASN is outside the permitted range.	CPX_INTERNAL_ERR
X'0100'	X'0001'	One or more input graphic characters were replaced with a “SUB” control character specified for the output string.	CPX_ADL_EXCEPTION_SEV3 (ADL_INV_IN_STRING)
X'0100'	X'0002'	One or more input graphic caracters were replaced with another graphic character or characters in CCSID2.	CPX_ADL_EXCEPTION_SEV3 (ADL_INV_IN_STRING)
X'0800'	X'0001'	Semaphore problem.	
X'0800'	X'0002'	CDRA dynamic link module problem occurred during initialization.	CPX_RESOURCE_LIM

DD&C Messages

The following messages are returned by the Parse and Generate functions. Two types of messages are produced:

1. Severe errors.

   These errors are either the result of calling a function with an incorrect set of parameters, or internal errors.

2. Information, warning, and error messages.

   These are errors of an internal nature. Because the ADLDCLSPC and ADLPLNSPC objects are usually created by the Parse Function, internal messages are more likely to occur at Parse time. Internal errors can occur, however, when the ADLDCLSPC or ADLPLNSPCs are not formed correctly, or when the Generate function does not work correctly. If you use the Generate function correctly and messages of this type still occur, resolving the error requires knowledge of the internal operation of the Generate function. Therefore, the message text and message explanation are often formulated specifically for IBM service personal.

Parse Function Messages

FMT1010I Start of include file *fn*.

Explanation: An INCLUDE statement has been successfully processed in the current ADL source file. The ADL Declaration Translator is starting to process the included file.

User Response: None.

FMT1011I End of include file *fn*.

Explanation: The ADL Declaration Translator has reached the end of the included ADL source file. It resumes processing the original ADL source file.

User Response: None.

FMT1201W System CCSID could not be obtained.

Explanation: For character literals, the current system CCSID is stored together with the literal in the ADLDCLSPC. An error occurred when trying to obtain the system CCSID. Therefore, character literals are stored with the CCSID value 0.

User Response: This is probably an internal error.

FMT1202W SKIP statement with length *n* inserted.

Explanation: In ADL, all data types other than ASIS and BIT must be byte-aligned. That is, before the current data declaration, a SKIP statement with an appropriate value is required so that the current data declaration begins on a byte boundary.

User Response: Check that the data declarations following the SKIP statement have the correct offsets.

FMT1203W SKIP attribute value changed from *n* to *m*.

Explanation: For arrays, each element of the array must be byte-aligned. To ensure this, an array declaration can have a SKIP attribute which defines the skip space necessary between two array elements.

User Response: Check the SKIP attribute produced and ensure that the declared array has the intended layout.

FMT1204W MAXLEN attribute specified but not evaluated.

Explanation: If a constant length is specified for fields of data types ASIS, BIT, and CHAR, then a MAXLEN attribute of the data type is ignored. A constant length for such a field can be specified, with the LENGTH attribute, for example.

User Response: If the ADL source is correct, no action is necessary.

FMT1205W DMNMAX attribute specified but not evaluated for this dimension.

Explanation: If the size of a dimension is fixed, then a DMNMAX attribute specified for this dimension is ignored.

User Response: If the ADL source is correct, no action is necessary.

FMT1207W Reference to constructor data type in LENGTH function could lead to declaration loop.

Explanation: When using the LENGTH function in declaration statements, it is possible to apply the LENGTH function to a data type which is not yet declared as in the following example:

```
a: CASE BEGIN;
   WHEN LENGTH(a)=16 then BIT LENGTH(8);
   OTHERWISE BIT LENGTH(16);
   END;
```

This warning message is issued to avoid a declaration loop in cases where such forward references occur.

User Response: Check that the data declaration referenced with the LENGTH function is completely declared before the current data declaration. In this case you can ignore the message.

FMT1400E Incorrect comment end symbol.

Explanation: The comment end symbol “*/” appeared in an ADL statement. This symbol is only allowed at the end of a comment. The symbol is ignored during subsequent parsing.

User Response: Remove the symbol.

FMT1401E Carriage return not allowed in bit literal.

Explanation: Only the characters 0 and 1 are allowed in bit literals.

User Response: Remove the carriage return.

FMT1402E Character *x* not allowed in bit literal.

Explanation: Only the characters 0 and 1 are allowed in bit literals.

User Response: Remove the character.

FMT1403E Carriage return not allowed in hexadecimal literal.

Explanation: Only the characters 0 to 9 and A to F inclusive are allowed in hexadecimal literals.

User Response: Remove the carriage return.

FMT1404E Character *x* not allowed in hexadecimal literal.

Explanation: Only the characters 0 to 9 and A to F inclusive are allowed in hexadecimal literals.

User Response: Remove the character.

FMT1405E Carriage return not allowed in character literal.

Explanation: Control characters are not allowed in character literals.

User Response: Either remove the character or use an encoded hexadecimal literal instead.

FMT1406E Carriage return not allowed in qualified identifier.

Explanation: Qualified identifiers must be written on one line.

User Response: Change the qualified identifier accordingly.

FMT1407E Identifier *id* is an ADL keyword.

Explanation: The identifier is not enclosed within double quotation marks ("). In some cases, this can lead to ambiguity.

User Response: Instead of the identifier BEGIN, for example, use "BEGIN".

FMT1408E Includes nested too deeply.

Explanation: The maximum number of nested INCLUDE statements allowed in ADL source files is 32.

User Response: The nesting level can be reduced by moving INCLUDE statements from lower-level to higher-level ADL source files. Ensure that such changes do not change the semantics of the ADL source, however.

FMT1409E File *fn* is already included.

Explanation: Including ADL source files results in a loop. The ADL Declaration Translator cannot resolve the INCLUDE statements for all open ADL source files.

User Response: Remove an INCLUDE statement for one ADL source file.

FMT1410E Unknown character *x* found in INCLUDE statement.

Explanation: The ADL Declaration Translator parsed the ADL keyword “INCLUDE”. Following this, it expects a character literal followed by a semicolon (;).

User Response: Remove the character.

FMT1411E Unexpected character *x* found.

Explanation: A character was detected during lexical analysis of the ADL source that is not allowed in this context.

User Response: Check the syntax of the ADL source at this point or earlier in the source for correctness.

FMT1412E File name missing for INCLUDE statement.

Explanation: The ADL keyword “INCLUDE” must be followed by a character literal identifying the file to include and then a semicolon (;) to end the INCLUDE statement.

User Response: Change the ADL source accordingly.

FMT1413E File *fn* not found.

Explanation: The file to be included could not be found.

User Response: Check that the file name is correct. If the environment variable ADLINC is used to search for the file to include, check the value of this variable.

FMT1414E Message text for *svr* message *msgno* not found.

Explanation: The ADL Declaration Translator attempted to issue a message. Due to an internal problem, the ADL Declaration Translator can only return the message severity and number, not the message text.

User Response: See the explanation for message number *msgno*.

FMT1415E Message *msg* is not in the range of declaration translator message numbers.

Explanation: The ADL Declaration Translator attempted to issue a message with an incorrect message number.

User Response: Search the ADL source at the specified point for a syntactic or semantic error.

FMT1416E Constant declaration for "*ref*" not found.

Explanation: A constant is referenced but not declared.

User Response: Either declare the constant or change the reference appropriately.

FMT1417E Subtype declaration for subtype instance "*sub*" not found.

Explanation: No subtype declaration is given for a subtype instance name.

User Response: Either declare the subtype or change the reference appropriately.

FMT1418E Constant declaration loop for "*ref*".

Explanation: The constant declaration references form a loop, for example:

```
a: CONSTANT b;
b: CONSTANT a;
```

User Response: Remove at least one reference from this declaration loop.

FMT1419E Subtype declaration loop for "*ref*".

Explanation: The subtype declaration references form a loop, for example:

```
a: SUBTYPE b;
b: SUBTYPE a;
```

User Response: Remove at least one subtype instance reference from this declaration loop.

FMT1420E A qualified identifier cannot be an output parameter.

Explanation: An output parameter is not allowed at this point in the ADL source. Only an input parameter, field of an input parameter record, or constant is allowed.

User Response: Change the ADL source accordingly.

FMT1421E A qualified identifier cannot be an input parameter.

Explanation: An input parameter is not allowed at this point in the ADL source. Only an output parameter, or field of an output parameter record, is allowed.

User Response: Change the ADL source accordingly.

FMT1422E Identifier with length *n* is longer than the allowed limit of *m*.

Explanation: There is a limit to the length of an ADL identifier.

User Response: Shorten the identifier accordingly.

FMT1423E Positional identifier "*n*" is too long.

Explanation: The maximum value of a positional identifier is defined as being “<max31>”.

User Response: Use an identifier instead of a positional identifier.

FMT1424E The literal with length *n* is too long.

Explanation: The maximum length allowed for a character literal is 32756 bytes.

User Response: Split up the literal into smaller segments.

FMT1425E A subtype or data type identifier exists with the same name.

Explanation: All data type, subtype, constant, WHEN statement, and OTHERWISE statement names must be unique within one DECLARE statement. A data type and a constant, for example, cannot have the same name.

User Response: Change or delete one of the affected names so that all names are unique.

FMT1426E A constant identifier exists with the same name.

Explanation: All data type, subtype, and constant names must be unique within one DECLARE statement. A data type and a constant, for example, cannot have the same name.

User Response: Change or delete one of the affected names so that all names are unique.

FMT1427E A DEFAULT statement already exists for the data type *typ*.

Explanation: Only one DEFAULT statement is allowed for each data type within a DECLARE statement.

User Response: Merge the affected statements into a single DEFAULT statement. If this is not possible, specify the attributes of the data type within the data declaration statement itself.

FMT1428E Identifier already used for a data declaration at the same level.

Explanation: Identifiers of data types need not be unique within a DECLARE statement, only at the current declaration level. If, for example, the identifier occurs as an element of a sequence declaration, then the names must be unique within this sequence.

User Response: Change or delete one of the affected names so that they are unique.

FMT1429E A DECLARE statement with the same name already exists.

Explanation: If DECLARE statements have names, then these names must be unique. In addition, no data type can have the same name as a DECLARE statement.

User Response: Change or delete one of the affected names so they are unique.

FMT1430E A PLAN statement with the same name already exists.

Explanation: All PLAN statements must have unique names.

User Response: Change or delete one of the affected names so they are unique.

FMT1431E The attribute has already been specified.

Explanation: Each attribute can be specified only once within an attribute list.

User Response: Delete one occurrence of the attribute.

FMT1432E Duplicate ENUMERATION value name.

Explanation: The names of ENUMERATION values must be unique.

User Response: Change or delete one of the affected names so they are unique.

FMT1433E Encoded hexadecimal literal has an uneven number of nibbles (*n*).

Explanation: Encoded hexadecimal literals for a given CCSID are interpreted by ADL as being character strings. Since each character contains at least two nibbles of 4 bits each, the encoded hexadecimal literal must have an even number of nibbles.

User Response: Add or subtract one nibble from the length of the literal.

FMT1434E Memory allocated to ADLDCLSPC with size *n* too small for ADLDCLSPC with size *m*.

Explanation: An ADL declare space is written to a block of memory that is allocated by the user. This error message occurs when the amount of memory allocated is insufficient. The ADL Declaration Translator writes the amount of space actually required into the first four bytes of the ADL declare space.

User Response: Call the ADL Declaration Translator again, allocating at least the specified amount of space to the ADL declare space.

FMT1435E Memory allocated to ADLPLNSPC with size *n* too small for ADLPLNSPC with size *m*.

Explanation: An ADL plan space is written to a block of memory that is allocated by the user. This error message occurs when the amount of memory allocated is insufficient. The ADL Declaration Translator writes the amount of space actually required into the first four bytes of the ADL plan space.

User Response: Call the ADL Declaration Translator again, allocating at least the specified amount of space to the ADL plan space.

FMT1436E Syntax error.

Explanation: While parsing ADL source, the parser could not apply a syntax rule to further process the ADL source. To resume parsing, subsequent lexical tokens are skipped until a token is identified from which parsing can resume. Following such an error, therefore, it cannot be guaranteed that the remaining ADL source text has been parsed correctly.

User Response: Check the ADL source text at or before the specified location for syntax errors.

FMT1437E Error at or before *n*, attribute "*tok*".

Explanation: The parser expected an attribute at the specified location. This error message occurred either because the current token could not be identified as an attribute, or because a syntax error occurred before the specified location.

User Response: Check that the specified token is a valid ADL attribute and that no syntax error occurred before this location.

FMT1438E *att* text too long.

Explanation: The length of the literal used to specify the value of the attribute is limited for some ADL attributes.

User Response: Reduce the length of the literal to within the permitted size.

FMT1439E Incorrect statement in *tok* statement.

Explanation: If the error occurred in a DECLARE statement, the parser expected one of the following statements at this location:

DEFAULT statement
CONSTANT statement
SUBTYPE statement
Data declaration statement.

If the error occurred in a PLAN statement, the parser expected either an assignment statement or a CALL statement.

User Response: Check the ADL source text at or before the specified location for syntax errors.

FMT1440E Incorrect plan parameter.

Explanation: At this point, the parser expected either an input parameter or an output parameter of the plan.

User Response: Check the ADL source text at or before the specified location for syntax errors.

FMT1441E Qualified identifier is already declared as a plan parameter.

Explanation: Plan parameters must be unique within a plan parameter list.

User Response: Delete one occurrence of the plan parameter.

FMT1442E Another PLAN statement exists with the same name.

Explanation: If PLAN statements (assignment statements and CALL statements) have names, then the names must be unique within a PLAN statement.

User Response: Change or delete one of the affected names so that all names are unique.

FMT1443E The program name literal must begin with "<".

Explanation: In a CALL statement, the character literal containing the program name must have the following syntax:

```
<library_path_name><file_path_name>
```

Spaces are only allowed within the library name.

User Response: Change the literal accordingly.

FMT1444E The character ">" is missing from the program name literal.

Explanation: In the character literal identifying the library name and function name of a CALL statement, the ADL Declaration Translator parsed the library name and expected the terminating character “>”. This was not found. See message FMT1443E for a description of the format of the program name literal.

User Response: Change the literal accordingly.

FMT1445E The character "<" is missing from the program name literal.

Explanation: In the character literal denoting the library name and function name of a CALL statement, the ADL Declaration Translator parsed the library name, including the terminating character “>”. A “<” character must immediately follow this character. See message FMT1443E for a description of the format of the program name literal.

User Response: Change the literal accordingly.

FMT1446E In the program name literal, blanks are not allowed within the function name.

Explanation: In the character literal denoting the library name and function name of a CALL statement, blanks are not allowed within the function name. See message FMT1443E for a description of the format of the program name literal.

User Response: Change the literal accordingly.

FMT1447E The character ">**" is missing from the program name literal.**

Explanation: In the character literal denoting the library name and function name of a CALL statement, the ADL Declaration Translator parsed the function name and expected the terminating character “>”. This was not found. See message FMT1443E for a description of the format of the program name literal.

User Response: Change the literal accordingly.

FMT1448E A qualified identifier is not allowed for the attribute *att*.

Explanation: Qualified identifiers can only be specified as values for certain ADL attributes. For all other ADL attributes, including the one referred to in this message, only literals and constants are allowed as values.

User Response: Change the attribute value to a literal. If the attribute allows a constant as a value, a constant may also be used.

FMT1449E PRECISION must be greater than zero for this data declaration.

Explanation: Only BINARY data types with SIGNED(TRUE) and SCALE(2) can have PRECISION(0). All other numeric data types must have PRECISION greater than zero.

User Response: Change the data declaration accordingly.

FMT1450E UNITLEN must be either 8 or 16 for this data type.

Explanation: For character fields, UNITLEN specifies the length of a character in bits. Only the values 8 and 16 are supported by ADL.

User Response: Change the data declaration accordingly.

FMT1451E The data type of the value of *typ* is not allowed.

Explanation: Depending on the context, a number of different data types are allowed at this point in the source text. Integer literals may be allowed but not character literals, for example.

User Response: Change the data type used to one of the data types allowed at this point in the source text. See *SMARTdata UTILITIES A Data Language Reference for Data Description and Conversion.* for details of which data types are allowed.

FMT1452E This attribute value is not allowed.

Explanation: ADL defines the constant values allowed for each attribute. The value specified in the source text is not valid for the current attribute.

User Response: Correct the attribute value. See *SMARTdata UTILITIES A Data Language Reference for Data Description and Conversion* for details of the attributes allowed.

FMT1453E This attribute is not allowed with a *tok* object.

Explanation: ADL objects, such as data types, dimension list attributes, subtype instances, DEFAULT statements, input and output parameters, and declarations, can have attributes. For each of these objects, there is a list of valid attributes. Not all attributes allowed with a data type, for example, are allowed with a DEFAULT statement for the data type. The current attribute is not allowed with the specified object.

User Response: Remove the attribute from the source text.

FMT1454E This identifier is already used in a higher-level data declaration.

Explanation: The following rule applies to the identifiers of data declarations: no fully-qualified identifier can be identical to a partially-qualified identifier within a DECLARE statement. This rule has been violated, that is, there are two identical identifiers of data declaration statements, on different levels but within the same hierarchy of data declarations.

User Response: Rename one of the two identifiers.

FMT1455E The identifier must contain at least one character that is not a digit.

Explanation: An identifier cannot be made up entirely of digits. At certain points within an ADL source, an identifier composed entirely of digits could be interpreted as an integer literal.

User Response: Change the name of the identifier to contain at least one character that is not a digit.

FMT1456E The declaration of this qualified identifier cannot be found.

Explanation: A qualified identifier is used to reference a data type, but no data declaration for this qualified identifier was found within the current DECLARE statement.

User Response: Either change the current qualified identifier or modify a data declaration or subtype declaration so that the reference can be resolved.

FMT1457E Several data declarations were found as references for this qualified identifier.

Explanation: Every data declaration can be uniquely identified, provided that all field and constructor declarations have names and that a fully-qualified name is used.

User Response: Use a fully-qualified identifier to ensure that the reference is unique.

FMT1458E The data declaration of this identifier must have the attribute SCALE(0).

Explanation: The data declaration of this identifier is used as an attribute value. Variable attribute values can only be declared with data declarations having the attribute SCALE(0).

User Response: Either change the data declaration of the identifier or use a different attribute value at the point where the data declaration is referred to.

FMT1459E The data declaration of this identifier must have the attribute COMPLEX(FALSE).

Explanation: The data declaration of this identifier is used as an attribute value. Variable attribute values can only be declared with data declarations having the attribute COMPLEX(FALSE).

User Response: Either change the data declaration of the identifier or use a different attribute value at the point where the data declaration is referred to.

FMT1460E Positional identifiers are not allowed at this point.

Explanation: Positional identifiers that are part of a qualified identifier are only allowed in PLAN statements, not in DECLARE statements.

User Response: If the qualified identifier remains unique, remove the positional identifier from the qualified identifier. Otherwise, use a name instead of a positional identifier for the part of the DECLARE statement referenced.

FMT1461E SEQUENCE body element "*tok*" not valid.

Explanation: At this point, the ADL Declaration Translator expected a data declaration for another member of the SEQUENCE.

User Response: Either change the current location to a valid data declaration, or complete the SEQUENCE declaration correctly according to the ADL syntax.

FMT1462E Elements of this array are themselves declared as arrays.

Explanation: In ADL, an element of an array cannot itself be an array. This also applies to an array element that is a subtype instance and the subtype is declared as an array.

User Response: There are several ways of resolving this situation:

1. Replace the current construction by an array declaration, with the dimension equal to the sum of the dimensions of the first two array elements.
2. Declare the array element of the outer array as a SEQUENCE with the inner array as the only data declaration.

FMT1463E Boolean operand expected for this operator.

Explanation: Boolean operators are boolean literals, boolean fields, or operators returning a boolean value. The data type of the operand does not belong to this group.

User Response: Change the expression so that the operator and operand agree.

FMT1464E Combination of operands not valid.

Explanation: For each operator, the following rules apply:

- Only certain data types and literal types are allowed
- Only certain combinations of operands are allowed for an operator.

For further details, see the description of the operators in the *SMARTdata UTILITIES A Data Language Reference for Data Description and Conversion*.

User Response: Change the expression so that only valid combinations of operands appear.

FMT1465E The value of the PRECISION attribute is too large or too small.

Explanation: The ADL Declaration Translator currently supports BINARY numbers with a maximum length of 32 bits. The value of PRECISION would result in a larger representation. For details, see the description of the BINARY data type in the *SMARTdata UTILITIES A Data Language Reference for Data Description and Conversion*.

User Response: Change the data declaration accordingly.

FMT1466E The value of PRECISION and the LENGTH attribute are not compatible.

Explanation: The BINARY field must be long enough for the number to have the requested precision. For details, see the description of the BINARY data type in the *SMARTdata UTILITIES A Data Language Reference for Data Description and Conversion*.

User Response: Change the data declaration accordingly.

FMT1467E The value of the LENGTH attribute is not valid.

Explanation: A BOOLEAN data type can have a length of between 1 and 64 bits. The length specified is not within this range.

User Response: Change the data declaration accordingly.

FMT1468E PRECISION is too large for the specified FORM value.

Explanation: For each format of the FLOAT data type, specified with the FORM attribute and a given RADIX, there is a maximum allowed value of PRECISION. For details, see the description of the FLOAT data type in the *SMARTdata UTILITIES A Data Language Reference for Data Description and Conversion*.

User Response: Change the data declaration accordingly.

FMT1469E The specified value of the MAXLEN attribute is not valid.

Explanation: For each data type where MAXLEN can be specified, valid attribute values are limited.

User Response: Check the rules for the current data type given in the *SMARTdata UTILITIES A Data Language Reference for Data Description and Conversion* and change the MAXLEN attribute accordingly.

FMT1470E The SGNUNS attribute cannot be used with the attributes SGNMNS and SGNPLS.

Explanation: For the PACKED data type, either the attribute SGNUNS can be specified or the attributes SGNMNS and SGNPLS, but not both.

User Response: Delete either the SGNUNS attribute or the attributes SGNMNS and SGNPLS from the current statement.

FMT1471E The value of the PRECISION attribute of this data declaration is not valid.

Explanation: The *SMARTdata UTILITIES A Data Language Reference for Data Description and Conversion* specifies a range of valid values for the PACKED data type. The value specified in the ADL source is outside this range.

User Response: Change the attribute’s value.

FMT1472E The value of the SCALE attribute of this data declaration is not valid.

Explanation: The *SMARTdata UTILITIES A Data Language Reference for Data Description and Conversion* specifies a range of valid values for the PACKED data type. The value specified in the ADL source is outside this range.

User Response: Change the attribute's value.

FMT1473E The value of the SGNLOC attribute of this data declaration is not valid.

Explanation: The *SMARTdata UTILITIES A Data Language Reference for Data Description and Conversion* specifies a range of valid values for the PACKED and ZONED data types. The value specified in the ADL source is outside this range.

User Response: Change the attribute's value.

FMT1474E The SGNMNS and SGNPLS attributes can only be specified together.

Explanation: For the PACKED and ZONED data types, the attributes SGNMNS and SGNPLS can be specified with the data declaration itself, subtype instances, or DEFAULT statements. In each case, either both or none of these attributes must be specified.

User Response: Change the ADL source accordingly.

FMT1475E The values for SGNMNS and SGNPLS must be mutually exclusive.

Explanation: Each hexadecimal digit in the SGNMNS or SGNPLS attribute represents the code of either the minus sign or plus sign, respectively. Since the same code cannot represent both the minus sign and plus sign, each hexadecimal digit can only appear in the hexadecimal literal of one of the two attributes.

User Response: Change the values of the SGNMNS and SGNPLS attributes so that the hexadecimal digits of the two literals are mutually exclusive.

FMT1476E A CCSID attribute must be specified for this data type.

Explanation: For a ZONED data type, if SIGNED is TRUE and SGNLOC has the values LSTBYT or FRSBYT, then a CCSID must be specified and applies to the sign character.

User Response: Change the attribute values accordingly.

FMT1477E For ENUMERATION data types, LENGTH must be either 8, 16, or 32.

Explanation: The length of the ENUMERATION field must be one of the specified values.

User Response: Change the attribute value accordingly.

FMT1478E The enumeration value of this identifier is outside the allowed range.

Explanation: An ENUMERATION data type can have various lengths and can be either signed or unsigned. Depending on these attributes, the enumeration values must be within a specific range, specified with the ENUMERATION data type.

User Response: Change either the identifier value so that it fits within the range, or the LENGTH or SIGNED attributes of this data type.

FMT1479E Attribute *att* must be specified.

Explanation: The attribute must be specified for this combination of data type and attribute.

User Response: Either change the values of other attributes of this data type, or the attribute specified with this data type.

FMT1480E The value of the LENGTH attribute is not allowed.

Explanation: For ASIS and BIT data types, there are restrictions on the values of the LENGTH attribute.

User Response: Change the attribute value to a valid value.

FMT1481E Duplicate ENUMERATION value.

Explanation: The numeric values of enumeration names must be unique.

User Response: Change one of the affected enumeration values so that they are unique.

FMT1482E Comment start symbol "/*" not allowed within a comment.

Explanation: A comment starts with the string “/*” and ends with the string “*/”. As comments cannot be nested, the string “/*” is not allowed inside a comment.

User Response: Check that the current string “/*” starts a comment. If so, it could be that the comment end symbol of the previous comment is missing.

FMT1483E The LENGTH attribute is mutually exclusive with the HIGH and LOW attributes.

Explanation: For the data type CHAR, either the attribute LENGTH can be specified or the attributes HIGH and LOW, but not both.

User Response: Remove the LENGTH attribute or the HIGH and LOW attributes from the current statement.

FMT1484E The HIGH and LOW attributes can only be specified together.

Explanation: For CHAR data types, the attributes HIGH and LOW can be specified with the data declaration itself, with subtype instances, or with DEFAULT statements. In each case, either both or neither of these two attributes must be specified.

User Response: Change the ADL source accordingly.

FMT1485E The value of HIGH must be greater than or equal to the value of LOW.

Explanation: If both HIGH and LOW are specified for a CHAR field and both are constants, then for the CHAR field to have a positive length the value of the HIGH attribute cannot be less than the value of the LOW attribute.

User Response: Change the ADL source accordingly.

FMT1486E Data declaration has a variable length but is not at the end of the SEQUENCE.

Explanation: Variable-length data declarations are only allowed as the last field of a SEQUENCE. Since the current data declaration is not the last in the SEQUENCE, this causes an error.

User Response: Change the ADL source accordingly.

FMT1487E Either DMNHIGH or DMNSIZE must be specified.

Explanation: For each dimension of an array, either of the attributes DMNHIGH or DMNSIZE must be specified, but not both.

User Response: Change the ADL source accordingly.

FMT1488E The value of DMNHIGH must be greater than or equal to the value of DMNLOW-1.

Explanation: If both DMNHIGH and DMNLOW are specified for a dimension of an ARRAY and are both constant, then for the dimension to have a size greater than or equal zero, the value of the DMNHIGH attribute cannot be less than the value of the DMNLOW attribute minus 1.

User Response: Change the ADL source accordingly.

FMT1489E DMNMAX must be specified for this dimension.

Explanation: If the size of an array dimension is variable, then at least the maximum size must be specified with the DMNMAX attribute.

User Response: Either specify the DMNMAX attribute or change the values for DMNSIZE or DMNLOW and DMNHIGH to constant values.

FMT1490E ARRAY elements must have fixed lengths.

Explanation: An array consists of a number of elements, each of the same length. Therefore, the data declaration used to declare the array elements must have a fixed size. Variable-length fields can appear in this declaration if MAXALC(TRUE) is specified for these fields .

User Response: Change the ADL source accordingly.

FMT1491E The DMNLST attribute must be specified.

Explanation: An array declaration defines the size of the array. This is done using the DMNLST (dimension list) attribute to declare the dimensions of the array.

User Response: Add the DMNLST attribute to the array declaration.

FMT1492E Variable attribute *att* not allowed here.

Explanation: The value of some attributes (for example, LENGTH) must be either fixed or variable, depending on the data type they are used with. This message appears when the attribute value must be fixed for the current data type and attribute.

User Response: Change the attribute value to a literal or a constant, as appropriate.

FMT1493E Length *n* missing on SKIP statement.

Explanation: In ADL, data types other than ASIS and BIT must be byte-aligned. This means that an appropriate SKIP statement must be inserted before the current data declaration so that the current data declaration starts on a byte boundary.

User Response: Either insert a SKIP statement with an appropriate value in the ADL source, or specify the AUTOSKIP option.

FMT1494E Array dimensions are fixed but MAXALC(FALSE) is specified.

Explanation: For an array, MAXALC(FALSE) can only be specified if at least one dimension has a variable size. A dimension has a variable size if at least one of the attributes DMNLOW, DMNHIGH or DMNSIZE has a qualified identifier as its value.

User Response: Change the ADL source accordingly.

FMT1495E SKIP attribute must have value *n* instead of value *m*.

Explanation: Each element of an array must be byte-aligned. To ensure this, an array declaration can have a SKIP attribute that defines the skip space necessary between two array elements.

User Response: Either insert the skip attribute with the proposed value in the ADL source, or specify the AUTOSKIP option.

FMT1496E Only numeric constants are allowed for this attribute.

Explanation: In plans, attributes can have qualified identifiers, identifiers, or constant literals as values. Only numeric values are allowed as literals.

User Response: Change the ADL source accordingly.

FMT1497E CCSID value cannot be zero.

Explanation: An encoded hex literal or character literal is treated as a character string with a system-independent representation. To interpret this string correctly, a CCSID must be specified. The value zero for the CCSID attribute means that no CCSID is specified. This is an error.

User Response: Specify the correct CCSID value. For character literals, the CCSID attribute can be omitted. In this case, the parser sets the CCSID of the character literal to the system CCSID.

FMT1498E Integer value too large or too small.

Explanation: Internally, the parser represents integer values as 32-bit signed binary numbers. The smallest and largest number that can be represented are defined as <min31> and <max31> respectively.

User Response: Remove the integer value from the ADL source.

FMT1499E Maximum size of this data type is too large.

Explanation: The largest field size supported by DD&C is <max31> bits. It is specified with the UNITLEN attribute together with either the MAXLEN attribute or the LOW and HIGH attributes.

User Response: Change the MAXLEN, LOW, HIGH or UNITLEN attributes so that the field size is below this limit.

FMT1500E The value of the LENGTH attribute is not valid.

Explanation: A BINARY data type can have a length of between 1 and 32 bits. The length specified is not in this range.

User Response: Change the data declaration accordingly.

FMT1501E A variable attribute value is not allowed here.

Explanation: The attribute values of DEFAULT statements must be fixed. The CCSID value of literals must also be fixed. For the given attribute, however, a data type defines the value.

User Response: Change the attribute value to a fixed value. Another possibility with default attributes is to define a subtype with the variable attribute value.

Warning: If, for example, the underlying data type is a `CHAR LENGTH(a)` field with a variable `a`, the attribute value is updated (in case the CHAR field is a target field in an assignment). If the same variable `a` is also used as length reference for other target fields, this variable is updated there also.

FMT1502E The number of *pln* parameters defined exceeds the limit of 255.

Explanation: In a plan, a maximum of 255 input and 255 output parameters can be defined. The specified parameter exceeds this limit.

User Response: There are several ways to reduce the number of parameters:

- Declare a structured data type that replaces several plan parameters.
- Split up the plan into several smaller plans and use workspace variables to store information between the processing of one plan and the next.

FMT1503E LENGTH attribute value (*n* bits) smaller than maximum size (*m* bits) of this CASE.

Explanation: The LENGTH attribute can be specified to define a larger CASE structure size than actually needed by its variants. At the indicated point in the ADL source, at least one variant of the CASE has a larger size than the specified value of the LENGTH attribute. Both sizes are measured in bits.

User Response: Check the CASE and change the declaration so that the specified lengths are consistent.

FMT1504E *n* is larger than the allowed limit of 2,147,483,647 bits.

Explanation: The size of the ARRAY and SEQUENCE constructors is limited.

User Response: If you must declare such large structures, try to split them up into a number of parts, each described by a separate data declaration in the same DECLARE statement. In the case of ARRAY, the declaration of a subtype describing the array fields might be helpful.

FMT1505E Length of referenced data type is not yet defined.

Explanation: In ADL source files, it is not permitted to use the LENGTH function of a data declaration that is currently declared, or to use the LENGTH function of a data declaration which itself depends on the currently declared data declaration. Were this to be allowed, then a data declaration loop would occur.

User Response: In the ADL source, remove at least one LENGTH function referencing a constructor data type.

FMT1506E Reference into array field is ambiguous.

Explanation: A reference to a data declaration that is part of an array is only permitted if the reference is located in the same array. This reference is then valid and always refers to the data declaration of the current array element.

User Response: Change the reference to a constant reference or to a reference of a data declaration outside the array.

FMT1800S Memory could not be allocated.

Explanation: When processing ADL source text, the ADL Declaration Translator allocates memory dynamically. This error occurs when insufficient system resources are available.

User Response: Do the following to help prevent this problem:

1. Check and if necessary increase the amount of memory and swapping space available on your machine.
2. Split up the ADL input files so that they can be translated independently.
3. Restart the ADL Declaration Translator.

FMT1801S Parser stack overflow.

Explanation: When parsing ADL source, information is stored in an internal stack. If the ADL source contains very deeply nested syntax structures, this stack can overflow.

User Response: Reduce the syntactic complexity of the ADL source, for example, by defining subtypes for complex nested structures. Alternatively, recompile the program that calls the FMTPRS function specifying a larger stack size.

FMT1802S Requested size for memory allocation is *n*, maximum allowed size is *m*.

Explanation: If the ADL source contains large literals or the ADL source itself is large, an internal memory allocation limit can be exceeded.

User Response: If large literals are used, split them into several parts without concatenating them. If the ADL source is large, try to split it up.

FMT1803S Internal system error, function *fnc*, rc *rc*.

Explanation: An error occurred when using operating system resources.

User Response: The error may disappear after modifying the ADL source. The error may also disappear after a system restart.

FMT1804S Pointer to condition token is incorrect.

Explanation: When calling the ADL Parse function, the pFeedBack parameter must point to a memory location that is large enough to hold the condition token structure.

User Response: Change the call to the Parse function accordingly.

FMT1810S Cannot open ADL source file *fn*.

Explanation: The pchSrcFilNam parameter of the ADL Parse function specifies the ADL source file. This file could not be found or opened.

User Response: Check that the file can be accessed and is not being used by another program.

FMT1811S Internal error detected by lexical analyzer at point *n*.

Explanation: Lexical analysis is the first step the parser undertakes when processing the ADL source.

User Response: Check the ADL source at or before the specified source location for syntax errors. If the problem persists, try modifying the ADL source slightly.

Generate Function Messages

FMT2951I Some code points not found in symbolic dump

Explanation: Some structures could not be expanded into specific fields because the code point was not found.

FMT2952I ADLLITBOOL with incorrect boolean value.

Explanation: The boolean value is incorrectly coded in the space. Only the boolean values TRUE = 0xf1 and FALSE = 0xf0 are allowed.

FMT2953I Last structure exceeds space length.

Explanation: The last structure could not be included in the symbolic dump because it was not completely inside the space.

FMT2901W Offset *n*: next pointer in anonymous ADLCNS not NULL.

Explanation: In an anonymous ADLCNS, the next pointer must be NULL, so that it is ignored for further processing.

FMT2851E Offset *n*: pointer in structure exceeds space length.

Explanation: The structure contains a pointer to a location outside the space. Look at the dump for further information about the structure and to indicate which element of the structure is not valid. Messages about structures within this structure may not be correct.

FMT2852E Offset *n*: structure exceeds space length.

Explanation: The structure is too long; a part of the structure is not inside the space. Look at the dump for further information about the structure.

FMT2853E Offset *n*: length *m* of structure must be less than 32K bytes.

Explanation: The length field of the structure contains an incorrect value. The DDM architecture defines that the value of the length field must be less than 32K bytes. Messages about structures within this structure may not be correct.

FMT2854E Offset *n*: length *m* of structure does not conform with code point *cp*.

Explanation: The length field of the structure contains an incorrect value. Messages about structures within this structure may not be correct.

FMT2855E Offset *n*: code point *cp* of structure not found.

Explanation: The code point of this structure is not valid for ADL encodings. Messages about structures within this structure may not be correct.

FMT2857E Offset *n*: structure has NULL length.

Explanation: The length field of the structure contains a NULL value. Messages about structures within this structure may not be correct.

FMT2858E Offset *n*: structure with incorrect pointer *m*.

Explanation: A pointer in the structure points to a location that is not a valid structure.

FMT2859E Offset *n*: anonymous ADLCNS in constant declaration chain.

Explanation: There is a constant without an identifier in the constant declaration chain.

FMT2860E Offset *n*: structure *m* pointing to structure *o* at offset *p* is not ADLCNS (0x300B).

Explanation: At this point, there must be an ADLCNS structure.

FMT2861E Offset *n*: the ADLCNS pointing to ADLLITBIT or ADLLITHEX at offset *m* has an attribute other than ADLLENGTH, or no attribute.

Explanation: The ADLCNS structure must have an ADLLENGTH attribute, and no other attributes.

FMT2862E Offset *n*: the ADLLENGTH pointed to in ADLCNS at offset *m* has a value other than ADLLITINT at offset *p*

Explanation: The ADLLENGTH structure must have an ADLLITINT value.

FMT2863E Offset *n*x: the ADLLITINT pointed to in ADLLENGTH at offset *m* has value greater than length of ADLLITBIT or AD

Explanation: The ADLLENGTH structure contains the length of the ADLLIT... value. Therefore, its length must conform with the structure length of the ADLLIT...

FMT2864E Offset *n*: the ADLCNS structure pointing to ADLLITEH or ADLLITCHAR at offset *m* has an attribute other than ADLCCSID, or no attribute.

Explanation: The ADLCNS structure must have an ADLCCSID attribute, and no other attributes.

FMT2865E Offset *n*: the boolean value in the ADLLITBOOL structure is *x*, not 0xf1 or 0xf0.

Explanation: The boolean value is incorrectly coded in the space. Only the boolean values TRUE = 0xf1 and FALSE = 0xf0 are allowed.

FMT2866E Offset *n*: loop within space: structure with code point *cp* is in its own subtree.

Explanation: The structure points to itself, possibly indirectly through another structure.

FMT2867E Offset *n*: the ADLQLFID structure with NULL identifier has next pointer that is not NULL.

Explanation: The subscript list is encoded using a ADLQLFID structure with a NULL identifier. There can be no other qualified identifier within a subscript list.

FMT2868E Offset *n*: the structure pointing to subscript list at offset *m* is not ADLQLFID.

Explanation: The subscript list is encoded using a ADLQLFID structure with a NULL identifier. It can only be used at the end of a qualified identifier.

FMT2869E Offset *n*: ADLID structure is longer than 255 characters.

Explanation: An ADL identifier can have a maximum length of 255 characters

FMT2870E Dump output stopped; structure with NULL length found.

Explanation: The dump output cannot be completed. The beginning of the next structure could not be found because of a NULL length field.

FMT2871E Offset *n*: code point *cp* of structure pointed to in structure offset *o* not found.

Explanation: The code point of this structure is not valid for ADL encodings.

FMT2872E Offset *n*: unexpected NULL pointer in structure.

Explanation: The Generator expected a pointer field to point to another structure, but the pointer is NULL.

FMT2873E Offset *n*: pointer *s* in structure *m* points to NULL.

Explanation: The Generator expected a pointer field to point to another structure, but the pointer is NULL.

FMT2802S Message text for error message *msgno*, severity *x* not found.

Explanation: The parameter DclXlrId does not contain the identifier of the ADL Declaration Translator.

User Response: Specify the correct value for DclXlrId. The value can be specified with the define `ADLDECLTRANSLATOR`.

FMT2803S Message *msgno* is not in the range of Generate message numbers.

Explanation: The message number is incorrect.

FMT2809S Incorrect parameter pointing to condition token.

Explanation: The parameter pFeedBack, pointing to the condition token that contains return value information, is incorrect.

FMT2811S Space with code point *cp* is not ADLDCLSPC or ADLPLNSPC.

Explanation: The space pointed to by pAdlSpc is not an ADL Declare Space or an ADL Plan Space. The valid code points are 3035 for the ADL Declare Space and 3055 for ADL Plan Space.

FMT2812S Offset *n*: space probably incomplete.

Explanation: The space is probably incomplete, because the memory allocated to the space and filled by the ADL Declaration Translator was too small to hold the entire space.

Appendix A. DD&C Sample Programs

The following examples use C language to demonstrate how to prepare a planned conversion using DD&C for Windows. The sample programs and listings include:

1. ADL Source Input
2. C Source Code - which shows the use of the FMTPRS, FMTGEN and FMTCRCP APIs
3. A Parse Function Optional Listing File
4. A Generate Function Optional Listing File
5. The Generate Function Source Output File
6. C Source Code for the Conversion Plan Executor - which shows the use of the FMTCPXI, FMTCPXC, and FMTCPXT APIs
7. ADL Source Input for a User Exit
8. C Source Code for a User Exit.

The source code for these sample programs, and the accompanying make and definition files, are available in the ***/usr/lpp/ddc/samples*** directory of the drive where DD&C for Windows is installed.

Sample Programs Showing the Use of the Parse and Generate Functions

The following sample program passes an ADL source file to the Parse function, then calls the Generate function to create ADL source text.

1. ADL Source Input - SAMPLE.ADL

Figure 14 shows the ADL source statements that are used as input to the sample program.

```
/******************************************************************************/
/* PRODUCT   = Data Description and Conversion                                */
/*                                                                            */
/* SOURCE FILE NAME = SAMPLE.ADL                                              */
/*                                                                            */
/* This ADL file is used by the SAMPLE1.C program to create a conversion      */
/* plan.                                                                      */
/******************************************************************************/

/******************************************************************************/
/* Declare statements for Input data structure                                */
/******************************************************************************/
DECLARE BEGIN;         /* ADL */
  COBOLREC: SEQUENCE BEGIN;
    INITIALS: CHAR LENGTH(3)
              CCSID(500);
    NUMBER: PACKED PRECISION(5);
  END;
END;

/******************************************************************************/
/* Declare statements for Output data structure                               */
/******************************************************************************/
DECLARE BEGIN;         /* ADL */
  CREC: SEQUENCE BEGIN;
    INITIALS: CHARSFX MAXLEN(4)
              CCSID(437);
    NUMBER: BINARY PRECISION(15)
            BYTRVS (TRUE );
  END;
END;

/******************************************************************************/
/* Plan statements for COBOL_TO_C conversion plan                             */
/******************************************************************************/
COBOL_TO_C: PLAN( COBOLREC: INPUT,
                  CREC: OUTPUT )
  BEGIN;
    CREC <- COBOLREC;
  END;

/******************************************************************************/
/* Plan statements for C_TO_COBOL conversion plan                             */
/******************************************************************************/
C_TO_COBOL: PLAN( CREC: INPUT,
                  COBOLREC: OUTPUT )
  BEGIN;
    COBOLREC <- CREC;
  END;
```

Figure 14. SAMPLE.ADL - Input ADL Source Text

2. C Source Code - SAMPLE1.C

Figure 15 shows the C source code for the Sample1 program. It uses the Parse function, the Generate function, and the Conversion Plan Builder APIs.

```
#pragma title ("SAMPLE1")
/******************************************************************************/
/* PRODUCT   = Data Description and Conversion                                */
/*                                                                            */
/* SOURCE FILE NAME = Sample1.C                                               */
/*                                                                            */
/* DESCRIPTIVE NAME = ADL Declaration Translator and CPB sample               */
/*                                                                            */
/* FUNCTION =  This sample program performs the following functions:          */
/*                                                                            */
/*             1. Calls the parse function of the ADL declaration translator  */
/*                to compile ADL source text SAMPLE.ADL into the appropriate  */
/*                ADL declare and plan spaces.                                */
/*             2. Calls the generate function of the ADL declaration          */
/*                translator to reproduce the ADL source file SAMPLE.GEN.     */
/*             3. Calls the conversion plan builder using the declear and     */
/*                plan spaces created by the parse function.  This will       */
/*                create a conversion plan.  The conversion plan will be used */
/*                by follow-on programs to do actual conversions.  The        */
/*                conversion plan will be stores in the file SAMPLE.SPC       */
/*                                                                            */
/* INPUTS  =  SAMPLE.ADL:  File containing ADL text                           */
/*                                                                            */
/* OUTPUTS =  SAMPLE_P.LST: File containing listing output from PARSE         */
/*                          funtion (FMTPRS).                                 */
/*            SAMPLE_G.LST: File containing listing output from Generate      */
/*                          funtion (FMTGEN).                                 */
/*            SAMPLE.GEN:   File containing ADL statements created by the     */
/*                          Generate function (FMTGEN).                       */
/*            SAMPLE.SPC:   File  containing the conversion plan created by   */
/*                          the FMTCRCP (Create Conversion Plan) function.    */
/* NOTES =                                                                    */
/*                                                                            */
/*   DEPENDENCIES = This program was compiled on OS/2, AIX, Windows NT 3.51,  */
/*                  and Windows 95 systems using the IBM VisaulAge compilers. */
/*                  Changes may be required to the Windows and OS/2 header    */
/*                  statements below when using a different compiler.        */
/*                                                                            */
/*                  The DD&C path statements must be set according to the     */
/*                  Getting Started book that is part of the Cobol            */
/*                  documentation.                                            */
/*                                                                            */
/*   RESTRICTIONS = None                                                      */
/*                                                                            */
/* ENTRY POINTS = main()                                                      */
/******************************************************************************/
```

Figure 15 (Part 1 of 7). SAMPLE1.C - C Source Code

```
#pragma page ()
/******************************************************************************/
/* Header Files.                                                              */
/******************************************************************************/

/*----------------------------------------------------------------------------*/
/* The following code adds the Windows header if this program is compiled     */
/* under the Windows operating system.  The following lines can be deleted    */
/* if this is not being compiled under Windows NT or Windows 95               */
/*----------------------------------------------------------------------------*/
#if defined(__WINDOWS__)               /* If compiled using Windows compiler  */
#include <windows.h>                   /*    include Windows header file      */
#endif                                 /* End if statement                    */

/*----------------------------------------------------------------------------*/
/* The following code adds OS/2 header definitions if this program is         */
/* compiled under the OS/2 operating system.  The following lines can be      */
/* deleted if this is not being compiled under OS/2.                          */
/*----------------------------------------------------------------------------*/
#if defined(__OS2__)                   /* If compiled using OS/2 compiler     */
#define INCL_BASE                      /*   All of OS/2 Base                  */
#define INCL_NOPMAPI                   /*   No presentation manager functions */
#include <os2.h>                       /*   Include OS/2 header file          */
#endif                                 /* End if statement                    */

/*---------------- C Library Header ------------------------------------------*/

#include <stdio.h>
#include <memory.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

#pragma page ()
/******************************************************************************/
/* DDC global header file                                                     */
/******************************************************************************/
#define FMT_NO_LCF   FMT_NO_LCF        /* exclude the Low Level Conversion
                                          Functions function prototypes and
                                          their declarations                  */
#define FMT_NO_CPEX  FMT_NO_CPEX       /* exclude the Conversion Plan
                                          Executor function prototypes and
                                          their declarations                  */

#include  "fmt.h"
```

Figure 15 (Part 2 of 7). SAMPLE1.C - C Source Code

```
/******************************************************************************/
/* Define ADLDCLSPC and ADLPLNSPC buffer length                               */
/******************************************************************************/
#define  BUFLEN_ADLDCLSPC  64000
#define  BUFLEN_ADLPLNSPC  64000
#define  BUFLEN_CNVPLNSPC   5000


/******************************************************************************/
/*  MAIN function                                                             */
/******************************************************************************/
int main( )
{
  /****************************************************************************/
  /* Local Variable definitions.                                              */
  /****************************************************************************/
  FMTCTOK     FeedBack;                     /* DD&C Feed Back area            */
  PBYTE       pAdlDclSpc;                   /* Ptr to ADL Declare Space       */
  PBYTE       pAdlPlnSpc;                   /* Ptr to ADL Plan Space          */
  PBYTE       *ppAdlDclSpcList = 0;         /* Ptr to array that contains     */
                                            /*   addresses of declare spaces  */
  PBYTE       *ppDefaultAdlPlnSpcList = 0;  /* Ptr to array that contains     */
                                            /*   addresses of plan spaces     */
  PVOID       pCnvPlnSpc = 0;               /* Ptr to Conversion Plan Space   */
  FMTCNSTKN   Cnstkn;                       /* Consistency Token              */
  FMTADLCA    MyIsInfo;                     /* ADL communications area        */
  FILE        *CnvPlnSpcHandle;             /* Handle to file that will hold  */
                                            /*   conversion plan              */
  ULONG       ulSpcLen;                     /* Length of conversion plan      */

  /****************************************************************************/
  /* Get space for ADLDCLSPC and ADLPLNSPC.                                   */
  /****************************************************************************/
  pAdlDclSpc = (PBYTE) malloc(BUFLEN_ADLDCLSPC);
  pAdlPlnSpc = (PBYTE) malloc(BUFLEN_ADLPLNSPC);

  /****************************************************************************/
  /* Call PARSE function of ADL Declaration Translator for ADL source text    */
  /* to get ADLDCLSPC and ADLPLNSPC.                                          */
  /* Type Manager id is set to ADL.                                           */
  /* Note: Currently all CCSID's should be zero.                              */
  /****************************************************************************/

  FMTPRS (  ADLDECLTRANSLATOR,             // PBYTE    pbDclXlrId
            0,                             // FMTCCSID lParameterCCSID
            10,                            // LONG     lSrcFilNamLength
            "SAMPLE.ADL",                  // PCHAR    pchSrcFilNam
            0,                             // FMTCCSID lSrcFilCCSID
            8,                             // LONG     lDclXlrOptLength
```

Figure 15 (Part 3 of 7). SAMPLE1.C - C Source Code

```
          "AUTOSKIP",                    // PCHAR    pchDclXlrOpt
          4,                             // LONG     lLstOptLength
          "LIST",                        // PCHAR    pchLstOpt
          12,                            // LONG     lLstFilNamLength
          "SAMPLE_P.LST",                // PCHAR    pchLstFilNam
          BUFLEN_ADLDCLSPC,              // LONG     lADLDclSpcLength
          pAdlDclSpc,                    // PBYTE    pbADLDclSpc
          0,                             // FMTCCSID lADLDclSpcCCSID
          &Cnstkn,                       // PFMTCNSTKN pbADLDclSpcCNSTKN
          BUFLEN_ADLPLNSPC,              // LONG     lADLPlnSpcLength
          pAdlPlnSpc,                    // PBYTE    pbADLPlnSpc
          &FeedBack );                   // PFMTCTOK pFeedBack

/*****************************************************************************/
/* Check the Condition Token                                                */
/*****************************************************************************/

if ( FeedBack.Condition_ID.usMsgNo != PRS_NO_ERROR )
 {
   printf("Error in PARSE function.\n");
   printf("The Condition Token has the following contents:\n");

   printf("Message Severity %d Number %d\n", FeedBack.Condition_ID.usMsgSev,
                                             FeedBack.Condition_ID.usMsgNo);
   printf("Service Condition Case %d\n",     FeedBack.fCase);
   printf("Condition Severity     %d\n",     FeedBack.fSeverity);
   printf("Control                %d\n",     FeedBack.fControl);
   printf("Facility ID            %c%c%c\n", FeedBack.uchFacility_ID⅛0‘,
                                             FeedBack.uchFacility_ID⅛1‘,
                                             FeedBack.uchFacility_ID⅛2‘);
   printf("Instance Specific      %d\n\n",   FeedBack.pI_S_Info.ulAdlExId);

 } /* endif */
else
 {
   /**************************************************************************/
   /* Call GENERATE function of ADL Declaration Translator for ADLDCLSPC    */
   /* to get ADL Source text.                                               */
   /* This call is not necessary to create a conversion plan, it is mainly  */
   /* done for debugging of the PARSE function.                             */
   /* Type Manager id is set to ADL.                                        */
   /* Note: Currently all CCSID's should be zero.                           */
   /**************************************************************************/
   FMTGEN (  ADLDECLTRANSLATOR,             // PBYTE    pbDclXlrId
             0,                             // FMTCCSID lParameterCCSID
             0,                             // LONG     lDclXlrOptLength
             "",                            // PCHAR    pchDclXlrOpt
             pAdlDclSpc,                    // PBYTE    pbAdlSpc
             0,                             // FMTCCSID lAdlSpcCCSID
```

Figure 15 (Part 4 of 7). SAMPLE1.C - C Source Code

```
          10,                            // LONG     lSrcFilNamLength
          "SAMPLE.GEN",                  // PCHAR    pchSrcFilNam
          0,                             // FMTCCSID lSrcFilCCSID
          12,                            // LONG     lLstOptLength
          "LIST FLAG(I)",                // PCHAR    pchLstOpt
          12,                            // LONG     lLstFilNamLength
          "SAMPLE_G.LST",                // PCHAR    pchLstFilNam
          0,                             // FMTCCSID lLstFilCCSID
          &FeedBack);                    // PFMTCTOK pFeedback

/*************************************************************************/
/* Check the Condition Token                                             */
/*************************************************************************/

if ( FeedBack.Condition_ID.usMsgNo != GEN_NO_ERROR )
 {
   printf("Error in GENERATE function.\n");
   printf("The Condition Token has the following contents:\n");

   printf("Message Severity %d Number %d\n",FeedBack.Condition_ID.usMsgSev,
                                            FeedBack.Condition_ID.usMsgNo);
   printf("Service Condition Case %d\n",    FeedBack.fCase);
   printf("Condition Severity     %d\n",    FeedBack.fSeverity);
   printf("Control                %d\n",    FeedBack.fControl);
   printf("Facility ID            %c%c%c\n",FeedBack.uchFacility_ID⅛0‘,
                                            FeedBack.uchFacility_ID⅛1‘,
                                            FeedBack.uchFacility_ID⅛2‘);
   printf("Instance Specific      %d\n\n",  FeedBack.pI_S_Info.ulAdlExId);

 } /* endif */

/*************************************************************************/
/* Get space for conversion plan space (CNVPLNSPC )                      */
/*************************************************************************/
pCnvPlnSpc     = malloc( BUFLEN_CNVPLNSPC );
memset( pCnvPlnSpc, '0', BUFLEN_CNVPLNSPC );

/*************************************************************************/
/* Initialize ADL Communication Area                                     */
/*************************************************************************/
FeedBack.pI_S_Info.pAdlCommArea = &MyIsInfo;

/*************************************************************************/
/* Call Conversion Plan Builder                                          */
/*************************************************************************/
ppAdlDclSpcList = malloc( sizeof( PBYTE ) );
ppAdlDclSpcList⅛0‘ = pAdlDclSpc;

ppDefaultAdlPlnSpcList = malloc( sizeof(PBYTE) );
ppDefaultAdlPlnSpcList⅛0‘ = pAdlPlnSpc;
```

Figure 15 (Part 5 of 7). SAMPLE1.C - C Source Code

```
FMTCRCP(
         1,                        // ULONG    ulAdlDclSpcCount
         ppAdlDclSpcList,          // PBYTE    *ppAdlDclSpcList
         0,                        // ULONG    ulUserAdlPlnSpcCount
         NULL,                     // PBYTE    *ppUserAdlPlnSpcList
         1,                        // ULONG    ulDefaultAdlPlnSpcCount
         ppDefaultAdlPlnSpcList,   // PBYTE    *ppDefaultAdlPlnSpcList
         BUFLEN_CNVPLNSPC,         // ULONG    ulCnvPlnSpcLength
         pCnvPlnSpc,               // PVOID    pCnvPlnSpc
         0,                        // ULONG    ulFlagList
         &FeedBack );              // PFMTCTOK pFeedback

/************************************************************************/
/* Check the Condition Token                                            */
/************************************************************************/

if ( FeedBack.Condition_ID.usMsgNo != CPB_NO_ERROR )
 {
   printf("Error in Conversion Plan Builder.\n");
   printf("The Condition Token has the following contents:\n");

   printf("Message Severity %d Number %d\n",FeedBack.Condition_ID.usMsgSev,
                                            FeedBack.Condition_ID.usMsgNo);
   printf("Service Condition Case %d\n",    FeedBack.fCase);
   printf("Condition Severity     %d\n",    FeedBack.fSeverity);
   printf("Control                %d\n",    FeedBack.fControl);
   printf("Facility ID            %c%c%c\n",FeedBack.uchFacility_ID⅛0‘,
                                            FeedBack.uchFacility_ID⅛1‘,
                                            FeedBack.uchFacility_ID⅛2‘);

   /*********************************************************************/
   /* Check whether an ADL exception occurred. If ADL exception the ADL  */
   /* communication area is filled.                                      */
   /*********************************************************************/
   if ( FeedBack.Condition_ID.usMsgNo == CPB_ADL_EXCEPTION_SEV2 ||
        FeedBack.Condition_ID.usMsgNo == CPB_ADL_EXCEPTION_SEV3 )
    {
      printf("The ADL communication area has the following contents:\n" );

      printf("ADL exception:             %d\n", MyIsInfo.lExId );
      printf("Severity of ADL exception: %d\n", MyIsInfo.usSevCod );
                                     /* The Severity of the ADL          */
                                     /* exception has the same value as  */
                                     /* the message severity             */
                                     /* ( Feedback.Condition_ID.usMsgSev */

      printf("Name of processed plan:    %.255s\n",
                                                MyIsInfo.PlanId.uchData );
```

Figure 15 (Part 6 of 7). SAMPLE1.C - C Source Code

```
           printf("Number of processed PLAN statement: %d\n",
                                                     MyIsInfo.lPlanStmt );
           printf("Source identifier of processed assignment statement: %.255s\n"
                  , MyIsInfo.SrcFldId.uchData );
           printf("Target identifier of processed assignment statement: %.255s\n"
                  , MyIsInfo.TrgFldId.uchData );


         } /* endif */

      } /* endif */
     else
      {
        /**********************************************************************/
        /* Write conversion plan space into file                              */
        /**********************************************************************/
        CnvPlnSpcHandle = fopen( "SAMPLE.SPC","wb");

        ulSpcLen = *((PULONG)pCnvPlnSpc);  /* Get length of the space out of  */
                                           /* the first 4 Byte                */

        fwrite( pCnvPlnSpc, sizeof(CHAR), ulSpcLen , CnvPlnSpcHandle );
        fclose( CnvPlnSpcHandle );

      } /* endelse */

   } /* endelse */

  /****************************************************************************/
  /* Free allocated resources                                                 */
  /****************************************************************************/
  if ( pAdlDclSpc != NULL ) {
    free( pAdlDclSpc );
  } /* endif */

  if ( pAdlPlnSpc != NULL ) {
    free( pAdlPlnSpc );
  } /* endif */

  if ( pCnvPlnSpc != NULL ) {
    free( pCnvPlnSpc );
  } /* endif */

  if ( ppAdlDclSpcList != NULL) {
    free( ppAdlDclSpcList );
  } /* endif */

  if ( ppDefaultAdlPlnSpcList != NULL ) {
    free( ppDefaultAdlPlnSpcList );
  } /* endif */
  return 0;
}
```

Figure 15 (Part 7 of 7). SAMPLE1.C - C Source Code

3. Parse Function Optional Listing File - SAMPLE_P.LIST

Figure 16 shows the optional listing file that is produced by the Parse function. It contains the ADL source input and related error messages (if any errors are found).

```
(0:0): informational FMT1000: Data Description and Conversion for OS/2 Version 1.10
(0:0): informational FMT1001: ADL Parser (c) Copyright IBM Corp. 1994.
(0:0): informational FMT1002: All rights reserved.
<0001> /******************************************************************************/
<0002> /* PRODUCT   = Data Description and Conversion                                */
<0003> /*                                                                            */
<0004> /* SOURCE FILE NAME = SAMPLE.ADL                                              */
<0005> /*                                                                            */
<0006> /* This ADL file is used by the SAMPLE1.C program to create a conversion      */
<0007> /* plan.                                                                      */
<0008> /******************************************************************************/
<0009>
<0010> /******************************************************************************/
<0011> /* Declare statements for Input data structure                                */
<0012> /******************************************************************************/
<0013> DECLARE BEGIN;         /* ADL */
<0014>   COBOLREC: SEQUENCE BEGIN;
<0015>     INITIALS: CHAR LENGTH(3)
<0016>               CCSID(500);
<0017>     NUMBER: PACKED PRECISION(5);
<0018>   END;
<0019> END;
<0020>
<0021> /******************************************************************************/
<0022> /* Declare statements for Output data structure                               */
<0023> /******************************************************************************/
<0024> DECLARE BEGIN;         /* ADL */
<0025>   CREC: SEQUENCE BEGIN;
<0026>     INITIALS: CHARSFX MAXLEN(4)
<0027>               CCSID(437);
<0028>     NUMBER: BINARY PRECISION(15)
<0029>             BYTRVS (TRUE );
<0030>   END;
<0031> END;
<0032>
<0033> /******************************************************************************/
<0034> /* Plan statements for COBOL_TO_C conversion plan                             */
<0035> /******************************************************************************/
<0036> COBOL_TO_C: PLAN( COBOLREC: INPUT,
<0037>                   CREC: OUTPUT )
<0038>   BEGIN;
<0039>     CREC <- COBOLREC;
<0040>   END;
<0041>
```

Figure 16 (Part 1 of 2). SAMPLE_P.LST - Parser Listing File

```
<0042> /******************************************************************************/
<0043> /* Plan statements for C_TO_COBOL conversion plan                             */
<0044> /******************************************************************************/
<0045> C_TO_COBOL: PLAN( CREC: INPUT,
<0046>                   COBOLREC: OUTPUT )
<0047>   BEGIN;
<0048>     COBOLREC <- CREC;
<0049>   END;
```

Figure 16 (Part 2 of 2). SAMPLE_P.LST - Parser Listing File

4. Generate Function Optional Listing File - SAMPLE_G.LIST

The optional Generate function listing file shown in Figure 17 contains a formatted dump of the ADL Declare Space and related error messages.

```
 Symbolic dump of ADL-Space
 Space class  3035 ADLDCLSPC
       length 00000485


00000000:  len 000c   class: 004f         SPCANC  next 0000000c  free 00000000
0000000c:  len 0020   class: 3002     ADLDECLARE  next 0000002c  name 00000000  attributes 00000000  defaults 00000244
                                                                 constants 00000000  subtypes 00000000  of 000003bd
0000002c:  len 0020   class: 3002     ADLDECLARE  next 00000000  name 00000000  attributes 00000000  defaults 0000004c
                                                                 constants 00000000  subtypes 00000000  of 00000175
0000004c:  len 0014   class: 304f     ADLDEFAULT  next 00000060  name 00000000  attributes 000000c7  type 0000016f
00000060:  len 0014   class: 304f     ADLDEFAULT  next 00000000  name 00000000  attributes 00000074  type 000000c1
00000074:  len 000c   class: 3034       ADLCCSID  next 00000080  value 000000b9
00000080:  len 000c   class: 303e      ADLMAXALC  next 0000008c  value 000000b4
0000008c:  len 000c   class: 303b      ADLMAXLEN  next 00000098  value 000000ac
00000098:  len 000c   class: 300c      ADLUNITLEN  next 00000000  value 000000a4
000000a4:  len 0008   class: 3056      ADLLITINT  value 00000008
000000ac:  len 0008   class: 3056      ADLLITINT  value 00000001
000000b4:  len 0005   class: 305e     ADLLITBOOL  value TRUE
000000b9:  len 0008   class: 3056      ADLLITINT  value 00000000
000000c1:  len 0006   class: 000c         CODPNT  value  3013      ADLCHARSFX
000000c7:  len 000c   class: 3028      ADLBYTRVS  next 000000d3  value 0000016a
000000d3:  len 000c   class: 302c     ADLCOMPLEX  next 000000df  value 00000165
000000df:  len 000c   class: 302d       ADLCNSTR  next 000000eb  value 00000160
000000eb:  len 000c   class: 305f         ADLFIT  next 000000f7  value 00000158
000000f7:  len 000c   class: 3041   ADLPRECISION  next 00000103  value 00000150
00000103:  len 000c   class: 3043       ADLRADIX  next 0000010f  value 00000148
0000010f:  len 000c   class: 3044       ADLSCALE  next 0000011b  value 00000140
0000011b:  len 000c   class: 3023      ADLSGNCNV  next 00000127  value 00000138
00000127:  len 000c   class: 304a      ADLSIGNED  next 00000000  value 00000133
00000133:  len 0005   class: 305e     ADLLITBOOL  value TRUE
00000138:  len 0008   class: 3056      ADLLITINT  value 00000001
00000140:  len 0008   class: 3056      ADLLITINT  value 00000000
00000148:  len 0008   class: 3056      ADLLITINT  value 00000002
00000150:  len 0008   class: 3056      ADLLITINT  value 0000001f
00000158:  len 0008   class: 3056      ADLLITINT  value 00000000
00000160:  len 0005   class: 305e     ADLLITBOOL  value FALSE
00000165:  len 0005   class: 305e     ADLLITBOOL  value FALSE
0000016a:  len 0005   class: 305e     ADLLITBOOL  value FALSE
0000016f:  len 0006   class: 000c         CODPNT  value  3014       ADLBINARY
00000175:  len 001c   class: 301b    ADLSEQUENCE  next 00000000  name 00000191  attributes 00000000  when 00000000
                                                                 fldsiz 00000199of 000001a1
```

Figure 17 (Part 1 of 3). SAMPLE_G.LST - ADLDCLSPC and Messages from Generate

```
00000191:  len 0008   class: 303c           ADLID  value "CREC"
00000199:  len 0008   class: 303f       ADLFLDSIZ  value 00000030
000001a1:  len 0018   class: 3013      ADLCHARSFX  next 000001b9  name 00000208  attributes 00000214  when 00000000
                                                                                                    fldsiz 0000023c
000001b9:  len 0018   class: 3014       ADLBINARY  next 00000000  name 000001d1  attributes 000001db  when 00000000
                                                                                                    fldsiz 00000200
000001d1:  len 000a   class: 303c           ADLID  value "NUMBER"
000001db:  len 000c   class: 3041    ADLPRECISION  next 000001e7  value 000001f8
000001e7:  len 000c   class: 3028       ADLBYTRVS  next 00000000  value 000001f3
000001f3:  len 0005   class: 305e      ADLLITBOOL  value TRUE
000001f8:  len 0008   class: 3056       ADLLITINT  value 0000000f
00000200:  len 0008   class: 303f       ADLFLDSIZ  value 00000010
00000208:  len 000c   class: 303c           ADLID  value "INITIALS"
00000214:  len 000c   class: 303b       ADLMAXLEN  next 00000220  value 00000234
00000220:  len 000c   class: 3034        ADLCCSID  next 00000000  value 0000022c
0000022c:  len 0008   class: 3056       ADLLITINT  value 000001b5
00000234:  len 0008   class: 3056       ADLLITINT  value 00000004
0000023c:  len 0008   class: 303f       ADLFLDSIZ  value 00000020
00000244:  len 0014   class: 304f      ADLDEFAULT  next 00000258  name 00000000  attributes 00000367  type 000003b7
00000258:  len 0014   class: 304f      ADLDEFAULT  next 00000000  name 00000000  attributes 0000026c  type 00000361
0000026c:  len 000c   class: 302c      ADLCOMPLEX  next 00000278  value 0000035c
00000278:  len 000c   class: 302d        ADLCNSTR  next 00000284  value 00000357
00000284:  len 000c   class: 305f          ADLFIT  next 00000290  value 0000034f
00000290:  len 000c   class: 3041    ADLPRECISION  next 0000029c  value 00000347
0000029c:  len 000c   class: 3044        ADLSCALE  next 000002a8  value 0000033f
000002a8:  len 000c   class: 3046       ADLSGNLOC  next 000002b4  value 00000337
000002b4:  len 000c   class: 3047       ADLSGNMNS  next 000002c0  value 0000030a
000002c0:  len 000c   class: 3048       ADLSGNPLS  next 000002cc  value 000002dd
000002cc:  len 000c   class: 304a       ADLSIGNED  next 00000000  value 000002d8
000002d8:  len 0005   class: 305e      ADLLITBOOL  value TRUE
000002dd:  len 0014   class: 300b          ADLCNS  next 00000000  name 00000000  attributes 000002f1  value 00000305
000002f1:  len 000c   class: 302b       ADLLENGTH  next 00000000  value 000002fd
000002fd:  len 0008   class: 3056       ADLLITINT  value 00000001
00000305:  len 0005   class: 305b       ADLLITHEX  value "c0"
0000030a:  len 0014   class: 300b          ADLCNS  next 00000000  name 00000000  attributes 0000031e  value 00000332
0000031e:  len 000c   class: 302b       ADLLENGTH  next 00000000  value 0000032a
0000032a:  len 0008   class: 3056       ADLLITINT  value 00000001
00000332:  len 0005   class: 305b       ADLLITHEX  value "d0"
00000337:  len 0008   class: 3056       ADLLITINT  value 00000000
0000033f:  len 0008   class: 3056       ADLLITINT  value 00000000
00000347:  len 0008   class: 3056       ADLLITINT  value 0000000f
0000034f:  len 0008   class: 3056       ADLLITINT  value 00000000
00000357:  len 0005   class: 305e      ADLLITBOOL  value FALSE
0000035c:  len 0005   class: 305e      ADLLITBOOL  value FALSE
00000361:  len 0006   class: 000c          CODPNT  value  3017       ADLPACKED
00000367:  len 000c   class: 3034        ADLCCSID  next 00000373  value 000003af
00000373:  len 000c   class: 3001      ADLJUSTIFY  next 0000037f  value 000003a7
0000037f:  len 000c   class: 302b       ADLLENGTH  next 0000038b  value 0000039f
0000038b:  len 000c   class: 300c      ADLUNITLEN  next 00000000  value 00000397
00000397:  len 0008   class: 3056       ADLLITINT  value 00000008
0000039f:  len 0008   class: 3056       ADLLITINT  value 00000001
```

Figure 17 (Part 2 of 3). SAMPLE_G.LST - ADLDCLSPC and Messages from Generate

```
000003a7:  len 0008   class: 3056       ADLLITINT  value 00000000
000003af:  len 0008   class: 3056       ADLLITINT  value 00000000
000003b7:  len 0006   class: 000c          CODPNT  value  3011         ADLCHAR
000003bd:  len 001c   class: 301b     ADLSEQUENCE  next 00000000  name 000003d9  attributes 00000000  when 00000000
                                                                                         fldsiz 000003e5of 000003ed
000003d9:  len 000c   class: 303c           ADLID  value "COBOLREC"
000003e5:  len 0008   class: 303f       ADLFLDSIZ  value 00000030
000003ed:  len 0018   class: 3011         ADLCHAR  next 00000405  name 00000443  attributes 0000044f  when 00000000
                                                                                                    fldsiz 00000477
00000405:  len 0018   class: 3017       ADLPACKED  next 00000000  name 0000041d  attributes 00000427  when 00000000
                                                          fldsiz 0000043b
0000041d:  len 000a   class: 303c           ADLID  value "NUMBER"
00000427:  len 000c   class: 3041    ADLPRECISION  next 00000000  value 00000433
00000433:  len 0008   class: 3056       ADLLITINT  value 00000005
0000043b:  len 0008   class: 303f       ADLFLDSIZ  value 00000018
00000443:  len 000c   class: 303c           ADLID  value "INITIALS"
0000044f:  len 000c   class: 302b       ADLLENGTH  next 0000045b  value 0000046f
0000045b:  len 000c   class: 3034        ADLCCSID  next 00000000  value 00000467
00000467:  len 0008   class: 3056       ADLLITINT  value 000001f4
0000046f:  len 0008   class: 3056       ADLLITINT  value 00000003
00000477:  len 0008   class: 303f       ADLFLDSIZ  value 00000018
```

Figure 17 (Part 3 of 3). SAMPLE_G.LST - ADLDCLSPC and Messages from Generate

5. Generate ADL Source Output - SAMPLE.GEN

Figure 18 shows the ADL source output that is produced by the Generate function. This ADL source is generated from the encoded ADL statements in the ADL Declare Space object.

```
DECLARE
BEGIN; /* declare */
   /*defaults*/
   DEFAULT CHAR CCSID(0) JUSTIFY(0) LENGTH(1) UNITLEN(8);
   DEFAULT PACKED COMPLEX(FALSE) CONSTRAINED(FALSE) FIT(0) PRECISION(
      15) SCALE(0) SGNLOC(0) SGNMNS(x'D') SGNPLS(x'C') SIGNED(TRUE);

   COBOLREC: SEQUENCE                           /* size: 48 */
      BEGIN; /*sequence*/
         INITIALS: CHAR LENGTH(3) CCSID(500);   /* size: 24 */
         NUMBER: PACKED PRECISION(5);           /* size: 24 */
      END; /*sequence*/
END; /* declare */

DECLARE
BEGIN; /* declare */
   /*defaults*/
   DEFAULT BINARY BYTRVS(FALSE) COMPLEX(FALSE) CONSTRAINED(FALSE) FIT(
      0) PRECISION(31) RADIX(2) SCALE(0) SGNCNV(1) SIGNED(TRUE);
   DEFAULT CHARSFX CCSID(0) MAXALC(TRUE) MAXLEN(1) UNITLEN(8);

   CREC: SEQUENCE                               /* size: 48 */
      BEGIN; /*sequence*/
         INITIALS: CHARSFX MAXLEN(4) CCSID(437); /* size: 32 */
         NUMBER: BINARY PRECISION(15) BYTRVS(TRUE); /* size: 16 */
      END; /*sequence*/
END; /* declare */
```

Figure 18. SAMPLE.GEN - ADL Source Produced by the Generate Function

Sample Program Showing the Use of The Conversion Plan Builder

The following sample program shows how to use DD&C for Windows to execute a planned conversion.

6. C Source Code for the Conversion Plan Executor - SAMPLE2.C

SAMPLE2.C calls the Conversion Plan Executor to convert data according to the conversion plans created by the Conversion Plan Builder in SAMPLE1.

Figure 19 shows the C source code for the Sample2 program.

```
#pragma title ("SAMPLE2")
/******************************************************************************/
/*                                                                            */
/* PRODUCT   = Data Description and Conversion                                */
/*                                                                            */
/* SOURCE FILE NAME = Sample2.C                                               */
/*                                                                            */
/* DESCRIPTIVE NAME = Conversion Plan Executor sample                         */
/*                                                                            */
/* FUNCTION =  This sample program performs the following functions:          */
/*                                                                            */
/*             1. Calls the DD&C functions of the conversion plan executor to */
/*                convert data based on the conversion plans created by the   */
/*                SAMPLE1 program.                                            */
/*             2. Hex string C1C2C301234C will be converted with specified    */
/*                plan COBOL_TO_C into the hex string 414243000D204.          */
/*             3. Hex string 414243000D204 will be converted with the         */
/*                specified plan C_TO_COBOL into the hex string C1C2C301234C. */
/*             4. The result will be printed on standard output.              */
/*                                                                            */
/* INPUTS  =  SAMPLE.SPC:  File containing conversion plan.  This file was    */
/*                         created by the Sample1 program.                    */
/*                                                                            */
/* OUTPUTS =  The following will be printed to the standard output:           */
/*                                                                            */
/*            Converted value for plan COBOL_TO_C: 41424300d204               */
/*            Converted value for plan C_TO_COBOL: c1c2c301234c               */
/*                                                                            */
/* NOTES =                                                                    */
/*                                                                            */
/*   DEPENDENCIES = This program was compiled on OS/2, AIX, Windows NT 3.51,  */
/*                  and Windows 95 systems using the IBM VisaulAge compilers. */
/*                  Changes may be required to the Windows and OS/2 header    */
/*                  statements below when using a different compiler.         */
/*                                                                            */
/*                  The DD&C path statements must be set according to the    */
/*                  Getting Started book that is part of the Cobol            */
/*                  documentation.                                            */
/*                                                                            */
/*   RESTRICTIONS = None                                                      */
/*                                                                            */
/* ENTRY POINTS = main()                                                      */
/*                                                                            */
/*                                                                            */
/******************************************************************************/
```

Figure 19 (Part 1 of 9). SAMPLE2.C - C Source Code for the Conversion Plan Executor

```
#pragma page ()
/******************************************************************************/
/* Header Files.                                                              */
/******************************************************************************/

/*----------------------------------------------------------------------------*/
/* The following code adds the Windows header if this program is compiled     */
/* under the Windows operating system.  The following lines can be deleted    */
/* if this is not being compiled under Windows NT or Windows 95               */
/*----------------------------------------------------------------------------*/
#if defined(__WINDOWS__)               /* If compiled using Windows compiler  */
#include <windows.h>                   /*    include Windows header file      */
#endif                                 /* End if statement                    */

/*----------------------------------------------------------------------------*/
/* The following code adds OS/2 header definitions if this program is         */
/* compiled under the OS/2 operating system.  The following lines can be      */
/* deleted if this is not being compiled under OS/2.                          */
/*----------------------------------------------------------------------------*/
#if defined(__OS2__)                   /* If compiled using OS/2 compiler     */
#define INCL_BASE                      /*   All of OS/2 Base                  */
#define INCL_NOPMAPI                   /*   No presentation manager functions */
#include <os2.h>                       /*   Include OS/2 header file          */
#endif                                 /* End if statement                    */

/*---------------- C Library Header ------------------------------------------*/

#include <stdio.h>
#include <memory.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <io.h>

#pragma page ()
/******************************************************************************/
/* DD&C global header                                                         */
/******************************************************************************/
#define FMT_NO_DCLXLRIFC FMT_NO_DCLXLRIFC
                                       /* exclude the Declaration Translator
                                          and Generate function prototypes and
                                          their declarations                  */
#define FMT_NO_CPB   FMT_NO_CPB        /* exclude the Conversion Plan Builder
                                          function prototypes and
                                          their declarations                  */
#define FMT_NO_LCF   FMT_NO_LCF        /* exclude the Low Level Conversion
                                          Functions function prototypes and
                                          their declarations                  */

#include  "fmt.h"
```

Figure 19 (Part 2 of 9). SAMPLE2.C - C Source Code for the Conversion Plan Executor

```
/******************************************************************************/
/* Define length of input and output buffer                                   */
/******************************************************************************/
#define  BUFFER_LENGTH  6

/******************************************************************************/
/* Enumeration to identify the appropriate CPEX function                      */
/******************************************************************************/
enum Execute { INIT,
               CONVERT,
               TERM
};

/******************************************************************************/
/*                                                                            */
/* The function PrintCtok prints the condition token and the ADL              */
/* communication area after an error occured in a conversion plan executor    */
/* function.                                                                  */
/*                                                                            */
/******************************************************************************/
void PrintCtok( PFMTCTOK  pFeedBack, PFMTADLCA pMyIsInfo, enum Execute Type )
{

  switch ( Type )
    {
      case INIT    : printf("Error in Conversion Plan Executor Init.\n");
        break;
      case CONVERT : printf("Error in Conversion Plan Executor Convert.\n");
        break;
      case TERM    : printf("Error in Conversion Plan Executor Term.\n");
        break;
    } /* endswitch */

  printf("The Condition Token has the following contents:\n");

  printf("Message Severity %d Number %d\n",pFeedBack->Condition_ID.usMsgSev,
                                         pFeedBack->Condition_ID.usMsgNo);
  printf("Service Condition Case %d\n",    pFeedBack->fCase);
  printf("Condition Severity     %d\n",    pFeedBack->fSeverity);
  printf("Control                %d\n",    pFeedBack->fControl);
  printf("Facility ID            %c%c%c\n",pFeedBack->uchFacility_ID⅛0‘,
                                         pFeedBack->uchFacility_ID⅛1‘,
                                         pFeedBack->uchFacility_ID⅛2‘);
```

Figure 19 (Part 3 of 9). SAMPLE2.C - C Source Code for the Conversion Plan Executor

```
  /****************************************************************************/
  /* Check whether an ADL exception occurred.                                 */
  /****************************************************************************/
  if ( pFeedBack->Condition_ID.usMsgNo == CPX_ADL_EXCEPTION_SEV2 ||
       pFeedBack->Condition_ID.usMsgNo == CPX_ADL_EXCEPTION_SEV3 )
   {
     if ( Type == INIT )
      {
        printf("ADL exception         %d\n", pFeedBack->pI_S_Info.ulAdlExId );
      } /* endif */
     else
      {
        printf("The ADL communication area has the following contents:\n" );

        printf("ADL exception:            %d\n", pMyIsInfo->lExId );
        printf("Severity of ADL exception: %d\n", pMyIsInfo->usSevCod );
                                       /* The Severity of the ADL          */
                                       /* exception has the same value as  */
                                       /* the message severity             */
                                       /* ( Feedback.Condition_ID.usMsgSev */

        printf("Name of processed plan:    %.255s\n",
                                                  pMyIsInfo->PlanId.uchData );
        printf("Number of processed PLAN statement: %d\n",
                                                  pMyIsInfo->lPlanStmt );
        printf("Input data portion that caused the error: %.255s\n"
               , pMyIsInfo->InpErrDta.uchData );
        printf("Source identifier of processed assignment statement: %.255s\n"
               , pMyIsInfo->SrcFldId.uchData );
        printf("Target identifier of processed assignment statement: %.255s\n"
               , pMyIsInfo->TrgFldId.uchData );

      } /* endelse */
   } /* endif */

  return;

}


/******************************************************************************/
/*  MAIN function                                                             */
/******************************************************************************/
int main( )
{
```

Figure 19 (Part 4 of 9). SAMPLE2.C - C Source Code for the Conversion Plan Executor

```
/****************************************************************************/
/* Local Variable definitions.                                              */
/****************************************************************************/
FMTCTOK     FeedBack;                     /* DD&C Feed Back area            */
FMTADLCA    MyIsInfo;                     /* ADL communications area        */
PVOID       pCnvPlnSpc;                   /* Ptr to Conversion Plan Space   */
FILE        *CnvPlnSpcHandle;             /* Handle to file that will hold  */
                                          /*   conversion plan              */
ULONG       ulLength;                     /* Length of conversion plan      */
ULONG       ulCnvPlnSpcHdl;               /* Handle to conversion plan space*/
PBYTE       *ppInputData = 0;             /* Ptr to array that contains     */
                                          /*   address of input parameters  */
PBYTE       *ppOutputData = 0;            /* Ptr to array that contains     */
                                          /*   address of output parameters */
PBYTE       pInValue = 0;                 /* Ptr to Buffer for input value  */
PBYTE       pOutValue = 0;                /* Ptr to Buffer for output value */
CHAR        Buffer⅛ BUFFER_LENGTH * 2 ‘;  /* Length of buffer               */
CHAR        EBCDIC⅛‘ = { 0xC1,0xC2,0xC3,0x01,0x23,0x4C }; /* EBCDIC chars   */
CHAR        ASCII⅛‘ = { 0x41,0x42,0x43,0x00,0xD2,0x04 };  /* ASCII  chars   */
int         k;                            /* Variable used for couter       */

/****************************************************************************/
/* Read the conversion plan space from file SAMPLE.SPC                      */
/****************************************************************************/

CnvPlnSpcHandle = fopen( "SAMPLE.SPC","rb");
ulLength = _filelength( fileno( CnvPlnSpcHandle ));
pCnvPlnSpc = (PVOID)calloc( ulLength, sizeof(CHAR));

fread( pCnvPlnSpc, sizeof(CHAR), ulLength, CnvPlnSpcHandle );

fclose( CnvPlnSpcHandle );


/****************************************************************************/
/* Call Conversion Plan Executor Initialization                             */
/****************************************************************************/

FMTCPXI( pCnvPlnSpc,                    // PBYTE    pCnvPlnSpc
         &ulCnvPlnSpcHdl,               // PULONG   pulCnvPlnSpcHdl
         &FeedBack );                   // PFMTCTOK pFeedback
```

Figure 19 (Part 5 of 9). SAMPLE2.C - C Source Code for the Conversion Plan Executor

```
/****************************************************************************/
/* Check the Condition Token                                                */
/****************************************************************************/
if ( FeedBack.Condition_ID.usMsgNo != CPX_NO_ERROR )
 {
   PrintCtok( &FeedBack, NULL, INIT );

 } /* endif */
else
 {
   /*************************************************************************/
   /* Initialize ADL Communication Area                                     */
   /*************************************************************************/
   FeedBack.pI_S_Info.pAdlCommArea = &MyIsInfo;

   /*************************************************************************/
   /* Alloc input and output buffer                                         */
   /*************************************************************************/
   pInValue     = malloc( BUFFER_LENGTH );
   pOutValue    = malloc( BUFFER_LENGTH );
   ppInputData  = malloc( sizeof( PBYTE ) );
   ppOutputData = malloc( sizeof(PBYTE) );

   /*************************************************************************/
   /* Call Conversion Plan Executor Convert                                 */
   /* In this conversion the plan COBOL_TO_C is executed. The input data    */
   /* are :                                                                 */
   /*    "ABC"  in international EBCDIC format  -> '0xC1C2C3'               */
   /*    1234   in PACKED PRECISION(5) format   -> '0x01234C'               */
   /*                                                                       */
   /* The output data after conversion should be:                           */
   /*    "ABC"  in Latin PC Data format + suffix  -> '0x41424300'           */
   /*    1234   in BINARY bytereversed format     -> '0xD204'               */
   /*************************************************************************/

   memcpy( pInValue, EBCDIC, sizeof(EBCDIC));
   memset( pOutValue, 0, BUFFER_LENGTH );


   ppInputData⅛0‘ = pInValue;
   ppOutputData⅛0‘ = pOutValue;


   FMTCPXC(
             ulCnvPlnSpcHdl,              // ULONG    ulCnvPlnSpcHdl
             10,                          // ULONG    ulPlnNamLength
             "COBOL_TO_C",                // PCHAR    pPlnNam
             1,                           // ULONG    ulInputParmNum
             ppInputData,                 // PBYTE    *ppInputData
```

Figure 19 (Part 6 of 9). SAMPLE2.C - C Source Code for the Conversion Plan Executor

```
          1,                           // ULONG    ulOutputParmNum
          ppOutputData,                // PBYTE    *ppInputData
          &FeedBack );                 // PFMTCTOK pFeedBack

if ( FeedBack.Condition_ID.usMsgNo != CPX_NO_ERROR )
 {
   /**********************************************************************/
   /* An error occured. Print the condition token                        */
   /**********************************************************************/
   PrintCtok( &FeedBack, &MyIsInfo, CONVERT );

 } /* endif */
else
 {
   /**********************************************************************/
   /* Print the converted value                                          */
   /**********************************************************************/
   printf("Converted value for plan COBOL_TO_C: " );
   for( k=0; k < BUFFER_LENGTH ;k++)
   {
     printf("%02x",pOutValue⅛k‘);
   }
   printf(" \n");

 } /* endelse */

/*************************************************************************/
/* Call Conversion Plan Executor Convert                                 */
/* In this conversion the plan C_TO_COBOL is executed. The input data    */
/* are :                                                                 */
/*    "ABC"  in Latin PC Data format + suffix  -> '0x41424300'           */
/*    1234   in BINARY bytereversed format     -> '0xD204'               */
/*                                                                       */
/* The output data after conversion should be:                           */
/*    "ABC"  in international EBCDIC format  -> '0xC1C2C3'               */
/*    1234   in PACKED PRECISION(5) format   -> '0x01234C'               */
/*************************************************************************/

memcpy( pInValue, ASCII, sizeof(ASCII));
memset( pOutValue, 0, BUFFER_LENGTH );

ppInputData⅛0‘ = pInValue;
ppOutputData⅛0‘ = pOutValue;


FMTCPXC(
          ulCnvPlnSpcHdl,              // ULONG    ulCnvPlnSpcHdl
          10,                          // ULONG    ulPlnNamLength
          "C_TO_COBOL",                // PCHAR    pPlnNam
```

Figure 19 (Part 7 of 9). SAMPLE2.C - C Source Code for the Conversion Plan Executor

```
            1,                           // ULONG    ulInputParmNum
            ppInputData,                 // PBYTE   *apInputData
            1,                           // ULONG    ulOutputParmNum
            ppOutputData,                // PBYTE   *apInputData
            &FeedBack );                 // PFMTCTOK pFeedBack

  if ( FeedBack.Condition_ID.usMsgNo != CPX_NO_ERROR )
   {
     /**********************************************************************/
     /* An error occured. Print the condition token                        */
     /**********************************************************************/
     PrintCtok( &FeedBack, &MyIsInfo, CONVERT );

   } /* endif */
  else
   {
     /**********************************************************************/
     /* Print the converted value                                          */
     /**********************************************************************/
     printf("Converted value for plan C_TO_COBOL: " );
     for( k=0; k < BUFFER_LENGTH ;k++)
     {
       printf("%02x",pOutValue⅛k‘);
     }
   } /* endelse */



  /*************************************************************************/
  /* Call Conversion Plan Executor Termination                             */
  /*************************************************************************/

  FMTCPXT(
           ulCnvPlnSpcHdl,               // ULONG    ulCnvPlnSpcHdl
           &FeedBack );                  // PFMTCTOK pFeedBack


  /*************************************************************************/
  /* Check the Condition Token                                             */
  /*************************************************************************/
  if ( FeedBack.Condition_ID.usMsgNo != CPX_NO_ERROR )
   {
     PrintCtok( &FeedBack, NULL, TERM );
   } /* endif */


} /* endelse */
```

Figure 19 (Part 8 of 9). SAMPLE2.C - C Source Code for the Conversion Plan Executor

```
  /****************************************************************************/
  /* Free allocated resources                                                 */
  /****************************************************************************/

  if ( pCnvPlnSpc != NULL ) {
    free( pCnvPlnSpc );
  } /* endif */

  if ( pInValue != NULL ) {
    free( pInValue );
  } /* endif */

  if ( pOutValue != NULL ) {
    free( pOutValue );
  } /* endif */

  if ( ppInputData != NULL) {
    free( ppInputData );
  } /* endif */

  if ( ppOutputData != NULL) {
    free( ppOutputData );
  } /* endif */

  return 0;
}
```

Figure 19 (Part 9 of 9). SAMPLE2.C - C Source Code for the Conversion Plan Executor

Sample Program Showing a User Exit

The following two examples demonstrate the use of the DD&C for Windows User Exit.

7. ADL Source Input - USEREXIT.ADL

Figure 20 shows the ADL source statements that are used as input to the sample User Exit program.

```
/******************************************************************************/
/* PRODUCT   = Data Description and Conversion                                */
/* SOURCE FILE NAME = USEREXIT.ADL                                            */
/* ADL source file for the USER EXIT sample.                                  */
/******************************************************************************/
/******************************************************************************/
/* Declare statements for source data structure                               */
/******************************************************************************/
source: DECLARE
BEGIN;
   number_of_char: CONSTANT 10;
   input1:         CHAR LENGTH( number_of_char );
END;
/*----------------------------------------------------------------------------*/
/******************************************************************************/
/* Declare statements for target data structure                               */
/******************************************************************************/
target: DECLARE
BEGIN;
   output1: CHAR LENGTH( 20 );
   output2: CHAR LENGTH( 20 );
END;
/*----------------------------------------------------------------------------*/
/******************************************************************************/
/* Declare statements for conversion plan                                     */
/******************************************************************************/
user_exit_plan: PLAN ( input1:    INPUT,
                       output1:   OUTPUT,
                       output2:   OUTPUT )
BEGIN;
   CALL '<userexit><ConvertToUpperOrLowerCaseChar>'
                 (  input1,
                    LENGTH( input1 ),
                    TRUE,             /* Convert to upper case character.     */
                    output1,
                    LENGTH( output1 )
                 );
   CALL '<userexit><ConvertToUpperOrLowerCaseChar>'
                 (  input1,
                    LENGTH( input1 ),
                    FALSE,            /* Convert to lower case character.     */
                    output2,
                    LENGTH( output2 )
                 );
END;
/*----------------------------------------------------------------------------*/
```

Figure 20. USEREXIT.ADL - ADL Source Input for User Exit

8. C Source Code - USEREXIT.C

Figure 21 shows the C source code for the sample User Exit program. This is a user-provided program called from within the conversion plan to convert character strings from upper case to lower case, or lower case to upper case.

```
#pragma title ("USEREXIT")
/******************************************************************************/
/*                                                                            */
/* PRODUCT   = Data Description and Conversion for Windows                    */
/*                                                                            */
/* SOURCE FILE NAME = USEREXIT.C                                              */
/*                                                                            */
/* DESCRIPTIVE NAME = User Exit Sample                                        */
/*                                                                            */
/* FUNCTION = This user exit sample program can be called via the             */
/*            DDC for Windows user exit facility.                             */
/*            It converts any character string from upper case to lower case  */
/*            characters or from lower case to upper case characters,         */
/*            depending on a boolean parameter.                               */
/* NOTES =                                                                    */
/*                                                                            */
/*   DEPENDENCIES = Windows 95 or Windows NT 3.51                             */
/*                                                                            */
/*   RESTRICTIONS = None                                                      */
/*                                                                            */
/* ENTRY POINTS = ConvertToUpperOrLowerCaseChar                               */
/*                                                                            */
/******************************************************************************/

#pragma page ()

/*----------------------------------------------------------------------------*/
/* The following code adds the Windows header if this program is compiled     */
/* under the Windows operating system.  The following lines can be deleted    */
/* if this is not being compiled under Windows NT or Windows 95               */
/*----------------------------------------------------------------------------*/
#if defined(__WINDOWS__)               /* If compiled using Windows compiler  */
#include <windows.h>                   /*    include Windows header file      */
#endif                                 /* End if statement                    */

/*----------------------------------------------------------------------------*/
/* The following code adds OS/2 header definitions if this program is         */
/* compiled under the OS/2 operating system.  The following lines can be      */
/* deleted if this is not being compiled under OS/2.                          */
/*----------------------------------------------------------------------------*/
#if defined(__OS2__)                   /* If compiled using OS/2 compiler     */
#define INCL_BASE                      /*   All of OS/2 Base                  */
#define INCL_NOPMAPI                   /*   No presentation manager functions */
#include <os2.h>                       /*   Include OS/2 header file          */
#endif                                 /* End if statement                    */
```

Figure 21 (Part 1 of 3). USEREXIT.C - C Source Code for User Exit

```
/*---------------- C Library Header ------------------------------------------*/
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>

/*---------------- DDC Global Header -----------------------------------------*/
#define  FMT_NO_DCLXLRIFC FMT_NO_DCLXLRIFC    /* Exclude FMTB.H (DCLXLRIFC)   */
#define  FMT_NO_CPB       FMT_NO_CPB          /* Exclude FMTC.H (CPB)         */
#define  FMT_NO_CPEX      FMT_NO_CPEX         /* Exclude FMTD.H (CPEX)        */
#define  FMT_NO_LCF       FMT_NO_LCF          /* Exclude FMTF.H (LCF)         */
#define  FMT_NO_UTL       FMT_NO_UTL          /* Exclude FMTU.H (UTL)         */
#include "fmt.h"


VOID APIENTRY ConvertToUpperOrLowerCaseChar( LONG, VOID *⅛‘, PFMTCTOK );


/******************************************************************************/
/* User Exit Function (API)                                                   */
/******************************************************************************/
VOID APIENTRY ConvertToUpperOrLowerCaseChar( LONG lParamCount,
                                             VOID *pParameter⅛‘,
                                             PFMTCTOK pFeedBack )

/******************************************************************************/
/* Expected parameters:                                                       */
/*    lParamCount:   5                                                        */
/*    pParameter⅛0‘: Reference to the input charater field.                   */
/*    pParameter⅛1‘: Reference to the length of the input character field.    */
/*    pParameter⅛2‘: Reference to a boolen condition.                         */
/*    pParameter⅛3‘: Reference to output character field.                     */
/*    pParameter⅛4‘: Reference to the length of the output character field.   */
/*    pFeedBack:     Reference to feedback area (NO ERROR: usMsgSev == 0).   */
/******************************************************************************/

{
   /***************************************************************************/
   /* Declaration and initialization of variables.                            */
   /***************************************************************************/
   INT   i;
   INT   CharToConvert;
   PCHAR pImputCharField = (PCHAR)(pParameter⅛0‘);
   LONG  lByteSizeOfInputChar = *((PLONG)pParameter⅛1‘);
   BOOL  fToUpperCaseLetter = *((PBOOL)pParameter⅛2‘);
   PCHAR pOutputCharField = (PCHAR)(pParameter⅛3‘);
   LONG  lByteSizeOfOutputChar = *((PLONG)pParameter⅛4‘);

   pFeedBack->Condition_ID.usMsgSev = 0;
   pFeedBack->Condition_ID.usMsgNo = 0;
```

Figure 21 (Part 2 of 3). USEREXIT.C - C Source Code for User Exit

```
   /**************************************************************************/
   /* Check buffer sizes and parameter count                                 */
   /**************************************************************************/
   if ( ( lByteSizeOfInputChar <= lByteSizeOfOutputChar ) &&
        ( lParamCount == 5 ) )
   {

      if (fToUpperCaseLetter == TRUE)
      {
         /********************************************************************/
         /* Convert all lower case character to upper case character         */
         /********************************************************************/
         for (i = 0; i < lByteSizeOfInputChar; i++)
         {
            CharToConvert = (INT)(*pImputCharField++);
            CharToConvert = toupper( CharToConvert );
            *pOutputCharField++ = (CHAR)CharToConvert;

         } /* endfor */
      }
      else
      {
         /********************************************************************/
         /* Convert all upper case character to lower case character         */
         /********************************************************************/
         for (i = 0; i < lByteSizeOfInputChar; i++)
         {
            CharToConvert = (INT)(*pImputCharField++);
            CharToConvert = tolower( CharToConvert );
            *pOutputCharField++ = (CHAR)CharToConvert;

         } /* endfor */
      } /* endif */
   }
   else
   {
      /***********************************************************************/
      /* Set error conditions                                                */
      /***********************************************************************/
      pFeedBack->Condition_ID.usMsgSev = 3;
      pFeedBack->Condition_ID.usMsgNo = 2647;
   } /* endif */

}
```

Figure 21 (Part 3 of 3). USEREXIT.C - C Source Code for User Exit

9. Parse USEREXIT.ADL and Create a Conversion Plan

The following is a sample program to parse USEREXIT.ADL and create a conversion plan.

```
#pragma title ("Sample3")
/******************************************************************************/
/*                                                                            */
/* PRODUCT   = Data Description and Conversion                                */
/*                                                                            */
/* SOURCE FILE NAME = Sample3.C                                               */
/*                                                                            */
/* DESCRIPTIVE NAME = Userexit Sample driver program                          */
/*                                                                            */
/* FUNCTION =  This sample program performs the following functions:          */
/*                                                                            */
/*             1. Calls the parse function (FMTPRS) of the ADL declaration    */
/*                translator to parse the ADL source text USEREXIT.ADL into   */
/*                the appropriate ADL declare and plan spaces.                */
/*             2. Calls the conversion plan builder (FMTCRCP) to create       */
/*                conversion plan from parser generated declare and plan      */
/*                spaces                                                      */
/*             3. The conversion plan builder output (conversion plan space)  */
/*                stored in the file SAMPLE3.SPC.  (This conversion plan      */
/*                will be used by the SAMPLE4.C program to do an actual       */
/*                conversion.                                                 */
/*                                                                            */
/* INPUTS  =  USEREXIT.ADL:  File containing ADL text                         */
/*                                                                            */
/* OUTPUTS =  SAMPLE3.LST:   File containing listing output from FMTPRS       */
/*                           (Parse) funtion.                                 */
/*            SAMPLE3.SPC:   File  containing the conversion plan created by  */
/*                           the FMTCRCP (Create Conversion Plan) function.   */
/*                                                                            */
/* NOTES =                                                                    */
/*                                                                            */
/*   DEPENDENCIES = This program was compiled on OS/2, AIX, Windows NT 3.51,  */
/*                  and Windows 95 systems using the IBM VisaulAge compilers. */
/*                  Changes may be required to the Windows and OS/2 header    */
/*                  statements below when using a different compiler.         */
/*                                                                            */
/*                  The DD&C path statements must be set according to the     */
/*                  Getting Started book that is part of the Cobol            */
/*                  documentation.                                            */
/*                                                                            */
/*   RESTRICTIONS = None                                                      */
/*                                                                            */
/* ENTRY POINTS = main()                                                      */
/*                                                                            */
/*                                                                            */
/******************************************************************************/
```

Figure 22 (Part 1 of 7). SAMPLE3.C - Parse USEREXIT.ADL and Create a Conversion Plan

```
#pragma page ()
/******************************************************************************/
/* Header Files.                                                              */
/******************************************************************************/

/*----------------------------------------------------------------------------*/
/* The following code adds the Windows header if this program is compiled     */
/* under the Windows operating system.  The following lines can be deleted   */
/* if this is not being compiled under Windows NT or Windows 95               */
/*----------------------------------------------------------------------------*/
#if defined(__WINDOWS__)               /* If compiled using Windows compiler  */
#include <windows.h>                   /*    include Windows header file      */
#endif                                 /* End if statement                    */

/*----------------------------------------------------------------------------*/
/* The following code adds OS/2 header definitions if this program is         */
/* compiled under the OS/2 operating system.  The following lines can be     */
/* deleted if this is not being compiled under OS/2.                          */
/*----------------------------------------------------------------------------*/
#if defined(__OS2__)                   /* If compiled using OS/2 compiler     */
#define INCL_BASE                      /*   All of OS/2 Base                  */
#define INCL_NOPMAPI                   /*   No presentation manager functions */
#include <os2.h>                       /*   Include OS/2 header file          */
#endif                                 /* End if statement                    */

/*---------------- C Library Header ------------------------------------------*/

#include <stdio.h>
#include <memory.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

#pragma page ()
/******************************************************************************/
/* DDC global header file                                                     */
/******************************************************************************/
#include  "fmt.h"

/******************************************************************************/
/* Define ADLDCLSPC and ADLPLNSPC buffer length                               */
/******************************************************************************/
#define  BUFLEN_ADLDCLSPC  64000
#define  BUFLEN_ADLPLNSPC  64000
#define  BUFLEN_CNVPLNSPC   5000
```

Figure 22 (Part 2 of 7). SAMPLE3.C - Parse USEREXIT.ADL and Create a Conversion Plan

```
/******************************************************************************/
/* Enumeration to identify the appropriate CPEX function                      */
/******************************************************************************/
enum Execute { PARSE,
               GENERATE,
               INIT,
               CONVERT,
               TERM
};

/******************************************************************************/
/*                                                                            */
/* The function PrintCtok prints the condition token and the ADL              */
/* communication area after an error occured in a conversion plan executor    */
/* function.                                                                  */
/*                                                                            */
/******************************************************************************/
void PrintCtok( PFMTCTOK  pFeedBack, PFMTADLCA pMyIsInfo, enum Execute Type )
{

  switch ( Type )
    {
      case PARSE   : printf("Error in Conversion Parser.\n");
        break;
      case GENERATE: printf("Error in Conversion Generate.\n");
        break;
      case INIT    : printf("Error in Conversion Plan Executor Init.\n");
        break;
      case CONVERT : printf("Error in Conversion Plan Executor Convert.\n");
        break;
      case TERM    : printf("Error in Conversion Plan Executor Term.\n");
        break;
    } /* endswitch */

  printf("The Condition Token has the following contents:\n");

  printf("Message Severity %d Number %d\n",pFeedBack->Condition_ID.usMsgSev,
                                           pFeedBack->Condition_ID.usMsgNo);
  printf("Service Condition Case %d\n",    pFeedBack->fCase);
  printf("Condition Severity     %d\n",    pFeedBack->fSeverity);
  printf("Control                %d\n",    pFeedBack->fControl);
  printf("Facility ID            %c%c%c\n",pFeedBack->uchFacility_ID⅛0‘,
                                           pFeedBack->uchFacility_ID⅛1‘,
                                           pFeedBack->uchFacility_ID⅛2‘);
```

Figure 22 (Part 3 of 7). SAMPLE3.C - Parse USEREXIT.ADL and Create a Conversion Plan

```
  /******************************************************************************/
  /* Check whether an ADL exception occurred.                                 */
  /******************************************************************************/
  if ( pFeedBack->Condition_ID.usMsgNo == CPX_ADL_EXCEPTION_SEV2 ||
       pFeedBack->Condition_ID.usMsgNo == CPX_ADL_EXCEPTION_SEV3 )
   {
     if ( Type == INIT )
      {
        printf("ADL exception         %d\n", pFeedBack->pI_S_Info.ulAdlExId );
      } /* endif */
     else
      {
        printf("The ADL communication area has the following contents:\n" );

        printf("ADL exception:             %d\n", pMyIsInfo->lExId );
        printf("Severity of ADL exception: %d\n", pMyIsInfo->usSevCod );
                                       /* The Severity of the ADL          */
                                       /* exception has the same value as  */
                                       /* the message severity             */
                                       /* ( Feedback.Condition_ID.usMsgSev */

        printf("Name of processed plan:    %.255s\n",
                                                  pMyIsInfo->PlanId.uchData );
        printf("Number of processed PLAN statement: %d\n",
                                                  pMyIsInfo->lPlanStmt );
        printf("Input data portion that caused the error: %.255s\n"
               , pMyIsInfo->InpErrDta.uchData );
        printf("Source identifier of processed assignment statement: %.255s\n"
               , pMyIsInfo->SrcFldId.uchData );
        printf("Target identifier of processed assignment statement: %.255s\n"
               , pMyIsInfo->TrgFldId.uchData );

      } /* endelse */
   } /* endif */

  return;

}

/********************************************************************************/
/*  MAIN function                                                               */
/********************************************************************************/
int main( )
{
```

Figure 22 (Part 4 of 7). SAMPLE3.C - Parse USEREXIT.ADL and Create a Conversion Plan

```
/***************************************************************************/
/* Local Variable definitions.                                             */
/***************************************************************************/
FMTCTOK     FeedBack;                     /* DD&C Feed Back area           */
PBYTE       pAdlDclSpc;                   /* Ptr to ADL Declare Space      */
PBYTE       pAdlPlnSpc;                   /* Ptr to ADL Plan Space         */
PBYTE       *ppAdlDclSpcList = 0;         /* Ptr to array that contains    */
                                          /*   addresses of declare spaces */
PBYTE       *ppDefaultAdlPlnSpcList = 0;  /* Ptr to array that contains    */
                                          /*   addresses of plan spaces    */
PVOID       pCnvPlnSpc = 0;               /* Ptr to Conversion Plan Space  */
FMTCNSTKN   Cnstkn;                       /* Consistency Token             */
FMTADLCA    MyIsInfo;                     /* ADL communications area       */
FILE        *CnvPlnSpcHandle;             /* Handle to file that will hold */
                                          /*   conversion plan             */
ULONG       ulSpcLen;                     /* Length of conversion plan     */
CHAR        pszAdlFileName⅛‘       = "USEREXIT.ADL";  /* Userexit ADL file  */
CHAR        pszPListFileName⅛‘     = "SAMPLE3.LST";   /* Parse listing file */
CHAR        pszConvPlanSpaceFile⅛‘ = "SAMPLE3.SPC"; /* Conversion Plan File */

/***************************************************************************/
/* Get space for ADLDCLSPC and ADLPLNSPC.                                  */
/***************************************************************************/
pAdlDclSpc     = (PBYTE) malloc(BUFLEN_ADLDCLSPC);
pAdlPlnSpc     = (PBYTE) malloc(BUFLEN_ADLPLNSPC);

/***************************************************************************/
/* Call PARSE function of ADL Declaration Translator for ADL source text   */
/* to get ADLDCLSPC and ADLPLNSPC.                                         */
/* Type Manager id is set to ADL.                                          */
/* Note: Currently all CCSID's should be zero.                             */
/***************************************************************************/
FMTPRS (  ADLDECLTRANSLATOR,             // PBYTE    pbDclXlrId
          0,                             // FMTCCSID lParameterCCSID
          strlen(pszAdlFileName),        // LONG     lSrcFilNamLength
          pszAdlFileName,                // PCHAR    pchSrcFilNam
          0,                             // FMTCCSID lSrcFilCCSID
          8,                             // LONG     lDclXlrOptLength
          "AUTOSKIP",                    // PCHAR    pchDclXlrOpt
          4,                             // LONG     lLstOptLength
          "LIST",                        // PCHAR    pchLstOpt
          strlen(pszPListFileName),      // LONG     lLstFilNamLength
          pszPListFileName,              // PCHAR    pchLstFilNam
          BUFLEN_ADLDCLSPC,              // LONG     lADLDclSpcLength
          pAdlDclSpc,                    // PBYTE    pbADLDclSpc
          0,                             // FMTCCSID lADLDclSpcCCSID
          &Cnstkn,                       // PFMTCNSTKN pbADLDclSpcCNSTKN
          BUFLEN_ADLPLNSPC,              // LONG     lADLPlnSpcLength
          pAdlPlnSpc,                    // PBYTE    pbADLPlnSpc
          &FeedBack );                   // PFMTCTOK pFeedBack
```

Figure 22 (Part 5 of 7). SAMPLE3.C - Parse USEREXIT.ADL and Create a Conversion Plan

```
/****************************************************************************/
/* Check the Condition Token                                                */
/****************************************************************************/
if ( FeedBack.Condition_ID.usMsgNo != PRS_NO_ERROR )
 {
   PrintCtok( &FeedBack, NULL, PARSE );
 } /* endif */
else
 {
   /*************************************************************************/
   /* Get space for conversion plan space (CNVPLNSPC )                      */
   /*************************************************************************/
   pCnvPlnSpc     = malloc( BUFLEN_CNVPLNSPC );
   memset( pCnvPlnSpc, '0', BUFLEN_CNVPLNSPC );

   /*************************************************************************/
   /* Initialize ADL Communication Area                                     */
   /*************************************************************************/
   FeedBack.pI_S_Info.pAdlCommArea = &MyIsInfo;

   /*************************************************************************/
   /* Call Conversion Plan Builder                                          */
   /*************************************************************************/
   ppAdlDclSpcList = malloc( sizeof( PBYTE ) );
   ppAdlDclSpcList⅛0‘ = pAdlDclSpc;

   ppDefaultAdlPlnSpcList = malloc( sizeof(PBYTE) );
   ppDefaultAdlPlnSpcList⅛0‘ = pAdlPlnSpc;

   FMTCRCP(
            1,                       // ULONG    ulAdlDclSpcCount
            ppAdlDclSpcList,         // PBYTE    *ppAdlDclSpcList
            0,                       // ULONG    ulUserAdlPlnSpcCount
            NULL,                    // PBYTE    *ppUserAdlPlnSpcList
            1,                       // ULONG    ulDefaultAdlPlnSpcCount
            ppDefaultAdlPlnSpcList,  // PBYTE    *ppDefaultAdlPlnSpcList
            BUFLEN_CNVPLNSPC,        // ULONG    ulCnvPlnSpcLength
            pCnvPlnSpc,              // PVOID    pCnvPlnSpc
            0,                       // ULONG    ulFlagList
            &FeedBack );             // PFMTCTOK pFeedback

   /*************************************************************************/
   /* Check the Condition Token                                             */
   /*************************************************************************/
   if ( FeedBack.Condition_ID.usMsgNo != CPB_NO_ERROR )
    {
      PrintCtok( &FeedBack, NULL, GENERATE );
    } /* endif */
   else
    {
```

Figure 22 (Part 6 of 7). SAMPLE3.C - Parse USEREXIT.ADL and Create a Conversion Plan

```
        /**********************************************************************/
        /* Write conversion plan space into file                              */
        /**********************************************************************/
        CnvPlnSpcHandle = fopen( pszConvPlanSpaceFile,"wb");

        ulSpcLen = *((PULONG)pCnvPlnSpc);  /* Get length of the space out of  */
                                           /* the first 4 Byte                */

        fwrite( pCnvPlnSpc, sizeof(CHAR), ulSpcLen , CnvPlnSpcHandle );
        fclose( CnvPlnSpcHandle );

      } /* endelse */

   } /* endelse */

  /****************************************************************************/
  /* Free allocated resources                                                 */
  /****************************************************************************/
  if ( pAdlDclSpc != NULL ) {
    free( pAdlDclSpc );
  } /* endif */

  if ( pAdlPlnSpc != NULL ) {
    free( pAdlPlnSpc );
  } /* endif */

  if ( pCnvPlnSpc != NULL ) {
    free( pCnvPlnSpc );
  } /* endif */

  if ( ppAdlDclSpcList != NULL) {
    free( ppAdlDclSpcList );
  } /* endif */

  if ( ppDefaultAdlPlnSpcList != NULL ) {
    free( ppDefaultAdlPlnSpcList );
  } /* endif */

  return 0;
}
```

Figure 22 (Part 7 of 7). SAMPLE3.C - Parse USEREXIT.ADL and Create a Conversion Plan

10. User Exit Conversion Plan Executor

The following is a sample program that calls the Conversion Plan Executor to convert data according to the conversion plan created by SAMPLE.C. This plan calls the user exit function ConvertToUpperOrLowerCaseChar defined in USEREXIT.C.

```
#pragma title ("SAMPLE4")
/******************************************************************************/
/*                                                                            */
/* PRODUCT   = Data Description and Conversion                                */
/*                                                                            */
/* SOURCE FILE NAME = Sample4.C                                               */
/*                                                                            */
/* DESCRIPTIVE NAME = Userexit Conversion Plan Executor sample                */
/*                                                                            */
/* FUNCTION =  This sample program performs the following functions:          */
/*                                                                            */
/*             1. Calls the DD&C functions of the conversion plan executor to */
/*                convert data based on the conversion plans created by the   */
/*                SAMPLE3.C program.                                          */
/*             2. Charater string "ABCDEabcde" will be converted to all       */
/*                upper case charaters using the USEREXIT function            */
/*                ConverToUpperOrLowerCaseChar                                */
/*             3. Charater string "ABCDEabcde" will be converted to all       */
/*                lower case charaters using the USEREXIT function            */
/*                ConverToUpperOrLowerCaseChar                                */
/*             4. The result will be printed to the standard output           */
/*                                                                            */
/* INPUTS  =  SAMPLE3.SPC: File containing conversion plan.  This file was    */
/*                         created by the Sample3 program.                    */
/*                                                                            */
/* OUTPUTS =  The following will be printed to the standard output:           */
/*                                                                            */
/*            The input string is ABCDEabcde                                  */
/*            The string converted to upper case is ABCDEABCDE                */
/*            The string converted to lower case is abcdeabcde                */
/*                                                                            */
/* NOTES =                                                                    */
/*                                                                            */
/*   DEPENDENCIES = This program was compiled on OS/2, AIX, Windows NT 3.51,  */
/*                  and Windows 95 systems using the IBM VisaulAge compilers. */
/*                  Changes may be required to the Windows and OS/2 header    */
/*                  statements below when using a different compiler.         */
/*                                                                            */
/*                  The DD&C path statements must be set according to the    */
/*                  Getting Started book that is part of the Cobol            */
/*                  documentation.                                            */
/*                                                                            */
/*   RESTRICTIONS = None                                                      */
/* ENTRY POINTS = main()                                                      */
/*                                                                            */
/******************************************************************************/
```

Figure 23 (Part 1 of 7). SAMPLE4.C - Call Conversion Plan Executor

```
#pragma page ()
/******************************************************************************/
/* Header Files.                                                              */
/******************************************************************************/

/*----------------------------------------------------------------------------*/
/* The following code adds the Windows header if this program is compiled     */
/* under the Windows operating system.  The following lines can be deleted    */
/* if this is not being compiled under Windows NT or Windows 95               */
/*----------------------------------------------------------------------------*/
#if defined(__WINDOWS__)               /* If compiled using Windows compiler  */
#include <windows.h>                   /*    include Windows header file      */
#endif                                 /* End if statement                    */

/*----------------------------------------------------------------------------*/
/* The following code adds OS/2 header definitions if this program is         */
/* compiled under the OS/2 operating system.  The following lines can be      */
/* deleted if this is not being compiled under OS/2.                          */
/*----------------------------------------------------------------------------*/
#if defined(__OS2__)                   /* If compiled using OS/2 compiler     */
#define INCL_BASE                      /*   All of OS/2 Base                  */
#define INCL_NOPMAPI                   /*   No presentation manager functions */
#include <os2.h>                       /*   Include OS/2 header file          */
#endif                                 /* End if statement                    */

/*---------------- C Library Header ------------------------------------------*/

#include <stdio.h>
#include <memory.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <io.h>

#pragma page ()
/******************************************************************************/
/* DD&C global header                                                         */
/******************************************************************************/
#define FMT_NO_DCLXLRIFC FMT_NO_DCLXLRIFC
                                       /* exclude the Declaration Translator
                                          and Generate function prototypes and
                                          their declarations                  */
#define FMT_NO_CPB   FMT_NO_CPB        /* exclude the Conversion Plan Builder
                                          function prototypes and
                                          their declarations                  */
#define FMT_NO_LCF   FMT_NO_LCF        /* exclude the Low Level Conversion
                                          Functions function prototypes and
                                          their declarations                  */

#include  "fmt.h"
```

Figure 23 (Part 2 of 7). SAMPLE4.C - Call Conversion Plan Executor

```
/******************************************************************************/
/* Define length of input and output buffer                                   */
/******************************************************************************/
#define  MAX_INPUT_LENGTH  10
#define  MAX_OUTPUT_LENGTH 20

/******************************************************************************/
/* Enumeration to identify the appropriate CPEX function                      */
/******************************************************************************/
enum Execute { INIT,
               CONVERT,
               TERM
};

/******************************************************************************/
/*                                                                            */
/* The function PrintCtok prints the condition token and the ADL              */
/* communication area after an error occured in a conversion plan executor   */
/* function.                                                                  */
/*                                                                            */
/******************************************************************************/
void PrintCtok( PFMTCTOK  pFeedBack, PFMTADLCA pMyIsInfo, enum Execute Type )
{

  switch ( Type )
    {
      case INIT    : printf("Error in Conversion Plan Executor Init.\n");
        break;
      case CONVERT : printf("Error in Conversion Plan Executor Convert.\n");
        break;
      case TERM    : printf("Error in Conversion Plan Executor Term.\n");
        break;
      default      : printf("Error During conversion.\n");
        break;
    } /* endswitch */

  printf("The Condition Token has the following contents:\n");

  printf("Message Severity %d Number %d\n",pFeedBack->Condition_ID.usMsgSev,
                                           pFeedBack->Condition_ID.usMsgNo);
  printf("Service Condition Case %d\n",    pFeedBack->fCase);
  printf("Condition Severity     %d\n",    pFeedBack->fSeverity);
  printf("Control                %d\n",    pFeedBack->fControl);
  printf("Facility ID            %c%c%c\n",pFeedBack->uchFacility_ID⅛0‘,
                                           pFeedBack->uchFacility_ID⅛1‘,
                                           pFeedBack->uchFacility_ID⅛2‘);
```

Figure 23 (Part 3 of 7). SAMPLE4.C - Call Conversion Plan Executor

```
  /****************************************************************************/
  /* Check whether an ADL exception occurred.                                 */
  /****************************************************************************/
  if ( pFeedBack->Condition_ID.usMsgNo == CPX_ADL_EXCEPTION_SEV2 ||
       pFeedBack->Condition_ID.usMsgNo == CPX_ADL_EXCEPTION_SEV3 )
   {
     if ( Type == INIT )
      {
        printf("ADL exception         %d\n", pFeedBack->pI_S_Info.ulAdlExId );
      } /* endif */
     else
      {
        printf("The ADL communication area has the following contents:\n" );

        printf("ADL exception:            %d\n", pMyIsInfo->lExId );
        printf("Severity of ADL exception: %d\n", pMyIsInfo->usSevCod );
                                       /* The Severity of the ADL          */
                                       /* exception has the same value as  */
                                       /* the message severity             */
                                       /* ( Feedback.Condition_ID.usMsgSev */

        printf("Name of processed plan:    %.255s\n",
                                                  pMyIsInfo->PlanId.uchData );
        printf("Number of processed PLAN statement: %d\n",
                                                  pMyIsInfo->lPlanStmt );
        printf("Input data portion that caused the error: %.255s\n"
               , pMyIsInfo->InpErrDta.uchData );
        printf("Source identifier of processed assignment statement: %.255s\n"
               , pMyIsInfo->SrcFldId.uchData );
        printf("Target identifier of processed assignment statement: %.255s\n"
               , pMyIsInfo->TrgFldId.uchData );

      } /* endelse */
   } /* endif */

  return;

}


/******************************************************************************/
/*  MAIN function                                                             */
/******************************************************************************/
int main( )
{
```

Figure 23 (Part 4 of 7). SAMPLE4.C - Call Conversion Plan Executor

```
/****************************************************************************/
/* Local Variable definitions.                                              */
/****************************************************************************/
FMTCTOK     FeedBack;                     /* DD&C Feed Back area            */
FMTADLCA    MyIsInfo;                     /* ADL communications area        */
PVOID       pCnvPlnSpc;                   /* Ptr to Conversion Plan Space   */
FILE        *CnvPlnSpcHandle;             /* Handle to file that will hold  */
                                          /*   conversion plan              */
ULONG       ulLength;                     /* Length of conversion plan      */
ULONG       ulCnvPlnSpcHdl;               /* Handle to conversion plan space*/
PBYTE       ppInputData⅛1‘;               /* Pointer to input parameter     */
PBYTE       ppOutputData⅛2‘;              /* Pointer to output parameters   */
CHAR        pszInString⅛MAX_INPUT_LENGTH+1‘ = "ABCDEabcde"; /* Input String */
CHAR        pszOutString1⅛MAX_OUTPUT_LENGTH+1‘; /* 1st Output string buffer */
CHAR        pszOutString2⅛MAX_OUTPUT_LENGTH+1‘; /* 2nd Output string buffer */
CHAR        pszConvPlanSpaceFile⅛‘ = "SAMPLE3.SPC";  /* Conversion plan     */
                                                     /*    space File       */

/****************************************************************************/
/* Read the conversion plan space from file SAMPLE.SPC                      */
/****************************************************************************/

/* -- Open the conversion Plan File for read                             -- */
CnvPlnSpcHandle = fopen( pszConvPlanSpaceFile,"rb");

/* -- Determine length of conversion plan                                -- */
ulLength = _filelength( fileno( CnvPlnSpcHandle ));

/* -- Allocate space for Conversion Plan                                 -- */
pCnvPlnSpc = (PVOID)calloc( ulLength, sizeof(CHAR));

/* -- Read conversion plan file into allocated space                     -- */
fread( pCnvPlnSpc, sizeof(CHAR), ulLength, CnvPlnSpcHandle );

/* -- Close conversion plan file.                                        -- */
fclose( CnvPlnSpcHandle );


/****************************************************************************/
/* Call Conversion Plan Executor Initialization                             */
/****************************************************************************/
FMTCPXI( pCnvPlnSpc,                    // PBYTE    pCnvPlnSpc
         &ulCnvPlnSpcHdl,               // PULONG   pulCnvPlnSpcHdl
         &FeedBack );                   // PFMTCTOK pFeedback
```

Figure 23 (Part 5 of 7). SAMPLE4.C - Call Conversion Plan Executor

```
/***************************************************************************/
/* Check the Condition Token                                               */
/***************************************************************************/
if ( FeedBack.Condition_ID.usMsgNo != CPX_NO_ERROR )
 {
   PrintCtok( &FeedBack, NULL, INIT );

 } /* endif */
else
 {
   /************************************************************************/
   /* Initialize ADL Communication Area                                    */
   /************************************************************************/
   FeedBack.pI_S_Info.pAdlCommArea = &MyIsInfo;

   /************************************************************************/
   /* Zero out output string buffers                                       */
   /************************************************************************/
   memset( pszOutString1, 0, MAX_OUTPUT_LENGTH );
   memset( pszOutString2, 0, MAX_OUTPUT_LENGTH );

   /************************************************************************/
   /* Set pointers for input and output strings                            */
   /************************************************************************/
   ppInputData⅛0‘  = pszInString;
   ppOutputData⅛0‘ = pszOutString1;
   ppOutputData⅛1‘ = pszOutString2;

   /************************************************************************/
   /* Call Conversion Plan Executor Convert                                */
   /* In this conversion the plan user_exit_plan is executed. The input is: */
   /*   pszInString = "ABCDEabcde"                                         */
   /*                                                                      */
   /* The output data after conversion should be:                          */
   /*   pszOutString1 = "ABCDEABCDE"                                       */
   /*   pszOutString2 = "abcdeabcde"                                       */
   /************************************************************************/
   FMTCPXC(
             ulCnvPlnSpcHdl,             // ULONG    ulCnvPlnSpcHdl
             14,                         // ULONG    ulPlnNamLength
             "user_exit_plan",           // PCHAR    pPlnNam
             1,                          // ULONG    ulInputParmNum
             ppInputData,                // PBYTE    *ppInputData
             2,                          // ULONG    ulOutputParmNum
             ppOutputData,               // PBYTE    *ppInputData
             &FeedBack );                // PFMTCTOK pFeedBack

   if ( FeedBack.Condition_ID.usMsgNo != CPX_NO_ERROR )
    {
```

Figure 23 (Part 6 of 7). SAMPLE4.C - Call Conversion Plan Executor

```
        /*********************************************************************/
        /* An error occured. Print the condition token                       */
        /*********************************************************************/
        PrintCtok( &FeedBack, &MyIsInfo, CONVERT );

      } /* endif */
     else
      {
        /*********************************************************************/
        /* Print the converted strings                                       */
        /*********************************************************************/
        printf("\nThe input string is %s\n", pszInString);
        printf("The string converted to upper case is %s\n", pszOutString1);
        printf("The string converted to lower case is %s\n", pszOutString2);

      } /* endelse */

     /************************************************************************/
     /* Call Conversion Plan Executor Termination                            */
     /************************************************************************/
     FMTCPXT(ulCnvPlnSpcHdl,               /* ULONG    ulCnvPlnSpcHdl        */
             &FeedBack );                  /* PFMTCTOK pFeedBack             */

     /************************************************************************/
     /* Check the Condition Token                                            */
     /************************************************************************/
     if ( FeedBack.Condition_ID.usMsgNo != CPX_NO_ERROR )
      {
        PrintCtok( &FeedBack, NULL, TERM );
      } /* endif */

   } /* endelse */

  /***************************************************************************/
  /* Free allocated resources                                                */
  /***************************************************************************/
  if ( pCnvPlnSpc != NULL ) {
    free( pCnvPlnSpc );
  } /* endif */

  /***************************************************************************/
  /* Exit Program                                                            */
  /***************************************************************************/
  return 0;
}
```

Figure 23 (Part 7 of 7). SAMPLE4.C - Call Conversion Plan Executor

Appendix B. Sample Programs in COBOL

The following examples use the COBOL language to demonstrate how to prepare a planned conversion using DD&C.

Note that all COBOL comment lines must begin in column 7.

```
****************************************************************
*  PRODUCT   = Data Description and Conversion                 *
*                                                              *
*  SOURCE FILE NAME = USEREXIT.CBL                             *
*                                                              *
*  DESCRIPTIVE NAME = User Exit Sample                         *
*                                                              *
*  FUNCTION = This user exit sample program can be called via  *
*              the DDC user exit facility.                     *
*              It converts any character string from upper case*
*              to lower case characters or from lower case to  *
*              upper case characters, depending on a boolean   *
*              parameter.                                      *
*                                                              *
*  ENTRY POINT = ConvertToUpperLowerCaseChar                   *
*                                                              *
****************************************************************
   IDENTIFICATION DIVISION.
   PROGRAM-ID. ConvertToUpperLowerCaseChar.
   ENVIRONMENT DIVISION.
   CONFIGURATION SECTION.
   SOURCE-COMPUTER. IBM-PS2.
   OBJECT-COMPUTER. IBM-PS2.
   DATA DIVISION.
   WORKING-STORAGE SECTION.
****************************************************************
*  Conversion tables and local variables                       *
****************************************************************
   01  UpperCase VALUE IS "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
       05 UpperChar PICTURE X(1) OCCURS 26 TIMES
                    INDEXED BY J.
   01  LowerCase  VALUE IS "abcdefghijklmnopqrstuvwxyz".
       05 LowerChar PICTURE X(1) OCCURS 26 TIMES
                    INDEXED BY L.
   77  Switch  PICTURE 9(6) USAGE IS BINARY.
   77  ADLTrue  PICTURE X(1) VALUE IS X"01".
   LINKAGE SECTION.
```

Figure 24 (Part 1 of 4). COBOL User Exit Sample

```
****************************************************************
*  Expected parameters:                                        *
*     ParamCount:      5                                       *
*     ParameterPtr(1)  Reference to the input character field. *
*     ParameterPtr(2)  Reference to the length of the input    *
*                      character field.                        *
*     ParameterPtr(3)  Reference to a boolean condition.       *
*     ParameterPtr(4)  Reference to output character field.    *
*     ParameterPtr(5)  Reference to the length of output       *
*                      character field.                        *
*     ParameterPtr(6)  Reference to feedback area              *
*                      (NO ERROR: MsgSev = 0).                 *
****************************************************************
   77  ParamCount  PICTURE 9(6) USAGE IS BINARY.
   01  ParameterList.
       05  ParameterPtr  USAGE IS POINTER OCCURS 5 TIMES.
   01  FeedBack.
       05 Condition-ID.
           10 MsgSev  PICTURE 9(2) USAGE IS BINARY.
           10 MsgNo    PICTURE 9(2) USAGE IS BINARY.
       05 Case-Severity-Control  PICTURE X.
       05 Facility-ID  PICTURE X(3).
       05 I-S-Info.
           10 AdlExId  PICTURE 9(6) USAGE IS BINARY.
           10 AdlCommAreaPtr REDEFINES AdlExId USAGE IS POINTER.
           10 User-ExitCtokPtr REDEFINES AdlExId USAGE IS POINTER.
   01  InputCharField.
       05 InputChar PICTURE X(1) OCCURS 1 TO 256 TIMES
                   DEPENDING ON ByteSizeOfInputChar
                   INDEXED BY I.
  77  ByteSizeOfInputChar  PICTURE 9(6) USAGE IS BINARY.
   77  ToUpperCaseLetter PICTURE X(1).
   01  OutputCharField.
       05 OutputChar PICTURE X(1) OCCURS 1 TO 256 TIMES
                    DEPENDING ON ByteSizeOfOutputChar
                    INDEXED BY K.
   77  ByteSizeOfOutputChar  PICTURE 9(6) USAGE IS BINARY.
   PROCEDURE DIVISION USING BY VALUE ParamCount
                            BY REFERENCE ParameterList
                            BY REFERENCE FeedBack.
****************************************************************
*  Initialization of variables                                 *
****************************************************************
       SET ADDRESS OF InputCharField TO ParameterPtr(1).
       SET ADDRESS OF ByteSizeOfInputChar TO ParameterPtr(2).
       SET ADDRESS OF ToUpperCaseLetter TO ParameterPtr(3).
       SET ADDRESS OF OutputCharField TO ParameterPtr(4).
       SET ADDRESS OF ByteSizeOfOutputChar TO ParameterPtr(5).
       MOVE 0 TO MsgSev OF Condition-ID IN FeedBack.
       MOVE 0 TO MsgNo OF Condition-ID IN FeedBack.
```

Figure 24 (Part 2 of 4). COBOL User Exit Sample

```
****************************************************************
*  Check buffer sizes and parameter count                      *
****************************************************************
       IF ByteSizeOfInputChar <= ByteSizeOfOutputChar AND
          ParamCount = 5
       THEN
          IF ToUpperCaseLetter = ADLTrue
          THEN
****************************************************************
*  Convert all lower case characters to upper case characters  *
****************************************************************
             PERFORM ToUpper VARYING I FROM 1 BY 1
                             UNTIL I > ByteSizeOfInputChar
          ELSE
****************************************************************
*  Convert all upper case characters to lower case characters  *
****************************************************************
             PERFORM ToLower VARYING I FROM 1 BY 1
                             UNTIL I > ByteSizeOfInputChar
       ELSE
****************************************************************
*  Set error conditions                                        *
****************************************************************
          MOVE 3 TO MsgSev OF Condition-ID IN FeedBack
          MOVE 2647 TO MsgNo OF Condition-ID IN FeedBack
       END-IF.
       EXIT PROGRAM.
   ToUpper.
       MOVE 0 TO Switch.
       SET K TO I.
       MOVE Inputchar(I) TO OutputChar(K).
       PERFORM VARYING J FROM 1 BY 1
               UNTIL J > 26 OR Switch = 1
          IF InputChar(I) = LowerChar(J)
       THEN
        MOVE UpperChar(J) TO OutputChar(K)
             MOVE 1 TO Switch
          END-IF
       END-PERFORM.
```

Figure 24 (Part 3 of 4). COBOL User Exit Sample

```
ToLower.
    MOVE 0 TO Switch.
    SET K TO I.
    MOVE Inputchar(I) TO OutputChar(K).
    PERFORM VARYING L FROM 1 BY 1
            UNTIL L > 26 OR Switch = 1
       IF InputChar(I) = UpperChar(L)
       THEN
          MOVE LowerChar(L) TO OutputChar(I)
          MOVE 1 TO Switch
       END-IF
    END-PERFORM.
END PROGRAM  ConvertToUpperLowerCaseChar.
```

Figure 24 (Part 4 of 4). COBOL User Exit Sample

ADL Declaration Translator and CPB Sample

```
****************************************************************
* PRODUCT = Data Description and Conversion                    *
*                                                              *
* SOURCE FILE NAME = Sample1.CBL                               *
*                                                              *
* DESCRIPTIVE NAME = ADL Declaration Translator and CPB sample *
*                                                              *
* FUNCTION = This sample program calls the parse function of   *
*        the ADL declaration translator to compile ADL source  *
*        text SAMPLE.ADL into the appropriate ADL declare      *
*        and plan spaces, calls the generate function of       *
*        the ADL declaration translator to reproduce the       *
*        ADL source file SAMPLE.GEN. The parse function's      *
*        output is also used to call the conversion plan       *
*        builder to create conversion plans from the encoded   *
*        descriptions.                                         *
*        The conversion plan space generated as the output     *
*        of the conversion plan builder is stored in the       *
*        file SAMPLEF.                                         *
*                                                              *
*****************************************************************
  IDENTIFICATION DIVISION.
  PROGRAM-ID. SAMPLE1.
  ENVIRONMENT DIVISION.
  CONFIGURATION SECTION.
  SOURCE-COMPUTER. IBM-PS2.
  OBJECT-COMPUTER. IBM-PS2.
  INPUT-OUTPUT SECTION.
  FILE-CONTROL.
       SELECT Sample-SPC ASSIGN TO SampleF
                                ORGANIZATION IS SEQUENTIAL.
  DATA DIVISION.
  FILE SECTION.
  FD   Sample-SPC
        RECORDING MODE IS F
        RECORD CONTAINS 5000 CHARACTERS.
  01    CnvPlnSpc  PICTURE X(5000).
  WORKING-STORAGE SECTION.
```

Figure 25 (Part 1 of 7). COBOL ADL Declaration Translator and CPB Sample

```
****************************************************************
* Work areas and values specific to this sample program        *
****************************************************************
  77     BUFLEN-ADLDCLSPC  PICTURE 9(6) USAGE IS BINARY
                           VALUE IS 64000.
  77     BUFLEN-ADLPLNSPC  PICTURE 9(6) USAGE IS BINARY
                           VALUE IS 64000.
  77     BUFLEN-CNVPLNSPC  PICTURE 9(6) USAGE IS BINARY
                           VALUE IS 5000.
  77     UserAdlPlnSpcNull  PICTURE 9(6) USAGE IS BINARY
                            VALUE IS 0.
  77     AdlDclSpc  PICTURE X(64000).
  77     AdlPlnSpc  PICTURE X(64000).
****************************************************************
*  Return codes and ADL exception codes                        *
****************************************************************
  77     PRS-NO-ERROR  PICTURE 9(2) USAGE BINARY VALUE IS 0.
  77     GEN-NO-ERROR  PICTURE 9(2) USAGE BINARY VALUE IS 0.
  77     CPB-NO-ERROR  PICTURE 9(2) USAGE BINARY VALUE IS 0.
  77     CPB-ADL-EXCEPTION-SEV2  PICTURE 9(6)
                                    USAGE IS BINARY VALUE is 2.
  77     CPB-ADL-EXCEPTION-SEV3  PICTURE 9(6)
                                    USAGE IS BINARY VALUE IS 18.
****************************************************************
* Parameters for FMTPRS, FMTGEN and FMTCRCP                    *
****************************************************************
  77     DclXlrId        PICTURE X(8).
  77     ADLDECLTRANSLATOR  PICTURE X(8)
                              VALUE IS X"2B12000301886D01".
  77     ParameterCCSID      PICTURE 9(6) USAGE IS BINARY
                             VALUE IS 0.
  77     SrcFilNamLength     PICTURE  9(6) USAGE IS BINARY.
  77     SrcFilNam           PICTURE  X(255).
  77     SrcFilCCSID         PICTURE  9(6) USAGE IS BINARY
                             VALUE IS 0.
  77     DclXlrOptLength     PICTURE  9(6) USAGE IS BINARY.
  77     DclXlrOpt           PICTURE  X(255).
  77     LstOptLength       PICTURE  9(6) USAGE IS BINARY.
  77     LstOpt              PICTURE  X(255).
  77     LstFilNamLength    PICTURE  9(6) USAGE IS BINARY.
  77     LstFilNam           PICTURE  X(255).
  77     ADLDclSpcCCSID      PICTURE  9(6)  USAGE IS BINARY
                             VALUE IS 0.
  77     ADLDclSpcLength   PICTURE  9(6) USAGE IS BINARY.
  77     ADLPlnSpcLength   PICTURE  9(6) USAGE IS BINARY.
  77     ADLSpcCCSID      PICTURE 9(6) USAGE IS BINARY
                          VALUE IS 0.
  77     LstFilCCSID         PICTURE 9(6) USAGE IS BINARY
                             VALUE IS 0.
```

Figure 25 (Part 2 of 7). COBOL ADL Declaration Translator and CPB Sample

```
  77     AdlDclSpcCount   PICTURE 9(6) USAGE IS BINARY.
  77     UserAdlPlnSpcCount   PICTURE  9(6) USAGE IS BINARY.
  77     DefaultAdlPlnSpcCount  PICTURE  9(6) USAGE IS BINARY.
  77     CnvPlnSpcLength   PICTURE 9(6) USAGE IS BINARY.
  77     FlagList      PICTURE 9(6) USAGE IS BINARY.
  01     AdlDclSpcList.
      05 AdlDclSpcptr   USAGE IS POINTER OCCURS 32 TIMES.
  01     UserAdlPlnSpcList.
      05 UserAdlPlnSpcPtr   USAGE IS POINTER OCCURS 32 TIMES.
  01     DefaultAdlPlnSpcList.
      05 DefaultAdlPlnSpcPtr  USAGE IS POINTER OCCURS 32 TIMES.
  01     Cnstkn.
      05 CnstknLength  PICTURE 9(6) USAGE IS BINARY.
      05 CnstknClass   PICTURE 9(2) USAGE IS BINARY.
      05 CnstknValue   PICTURE X(16).
  01     FeedBack.
      05 Condition-ID.
         10 MsgSev  PICTURE 9(2) USAGE IS BINARY.
         10 MsgNo    PICTURE 9(2) USAGE IS BINARY.
      05 Case-Severity-Control  PICTURE X.
      05 Facility-ID  PICTURE X(3).
      05 I-S-Info.
         10 AdlExId  PICTURE 9(6) USAGE IS BINARY.
         10 AdlCommAreaPtr REDEFINES AdlExId USAGE IS POINTER.
         10 User-ExitCtokPtr  REDEFINES AdlExId USAGE IS POINTER.
****************************************************************
* ADL Communication Area                                       *
****************************************************************
  01     AdlCommArea.
      05 AdlCALength  PICTURE 9(6) USAGE IS BINARY.
      05 ExId  PICTURE 9(6) USAGE IS BINARY.
      05 SevCod  PICTURE 9(2) USAGE IS BINARY.
      05 PlanId.
         10  PreLength  PICTURE 9(2) USAGE IS BINARY.
         10  CharData  PICTURE X(255).
      05 PlanStmt  PICTURE 9(6) USAGE IS BINARY.
      05 InpErrDta.
         10  PreLength  PICTURE 9(2) USAGE IS BINARY.
         10  CharData  PICTURE X(255).
      05 SrcFldID.
         10  PreLength  PICTURE 9(2) USAGE IS BINARY.
         10  CharData  PICTURE X(255).
      05 TrgFldId.
         10  PreLength  PICTURE 9(2) USAGE IS BINARY.
         10  CharData  PICTURE X(255).
  PROCEDURE DIVISION.
```

Figure 25 (Part 3 of 7). COBOL ADL Declaration Translator and CPB Sample

```
****************************************************************
* Call PARSE function of ADL Declaration Translator for        *
* ADL source text to get ADLDCLSPC and ADLPLNSPC.              *
* Translator id is set to ADL.                                 *
* Note: Currently all CCSIDs should be zero                    *
****************************************************************
       MOVE ADLDECLTRANSLATOR TO DclXlrId.
       MOVE 10 TO SrcFilNamLength.
       MOVE "SAMPLE.ADL" TO SrcFilNam.
       MOVE 8 TO DclXlrOptLength.
       MOVE "AUTOSKIP" TO DclXlrOpt.
       MOVE 4 TO LstOptLength.
       MOVE "LIST" TO LstOpt.
       MOVE 12 TO LstFilNamLength.
       MOVE "SAMPLE_P.LST" TO LstFilNam.
       MOVE BUFLEN-ADLDCLSPC TO ADLDclSpcLength.
       MOVE BUFLEN-ADLPLNSPC TO ADLPlnSpcLength.
       CALL "FMTPRS" USING
                            BY REFERENCE DclXlrId
                            BY VALUE     ParameterCCSID
                            BY VALUE     SrcFilNamLength
                            BY REFERENCE SrcFilNam
                            BY VALUE     SrcFilCCSID
                            BY VALUE     DclXlrOptLength
                            BY REFERENCE DclXlrOpt
                            BY VALUE     LstOptLength
                            BY REFERENCE LstOpt
                            BY VALUE     LstFilNamLength
                            BY REFERENCE LstFilNam
                            BY VALUE     ADLDclSpcLength
                            BY REFERENCE AdlDclSpc
                            BY VALUE     ADLDclSpcCCSID
                            BY REFERENCE Cnstkn
                            BY VALUE     ADLPlnSpcLength
                            BY REFERENCE AdlPlnSpc
                            BY REFERENCE FeedBack.
****************************************************************
* Check the Condition Token                                    *
****************************************************************
       IF MsgNo OF Condition-ID IN FeedBack NOT = PRS-NO-ERROR
       THEN
            DISPLAY "Error in PARSE function"
            DISPLAY
             "The Condition Token has the following contents:"
            DISPLAY "Message Severity "
                    MsgSev OF Condition-ID IN FeedBack
                    " Number "
                    MsgNo OF Condition-ID IN FeedBack
            DISPLAY "Case+Severity+Control "
                    Case-Severity-Control IN FeedBack
```

Figure 25 (Part 4 of 7). COBOL ADL Declaration Translator and CPB Sample

```
            DISPLAY "Facility ID "
                    Facility-ID OF FeedBack
            DISPLAY "Instance Specific "
                    AdlExId OF FeedBack
            STOP RUN
       END-IF.
****************************************************************
* Call GENERATE function of ADL Declaration Translator for     *
* ADLDCLSPC to get ADL Source test.                            *
* This call is not necessary to create a conversion plan.      *
* It is mainly done for debugging of the PARSE function.       *
* Translator id is set to ADL.                                 *
* Note: Currently all CCSIDs should be zero.                   *
****************************************************************
       MOVE 10 to SrcFilNamLength.
       MOVE "SAMPLE.GEN" TO SrcFilNam.
       MOVE 0 to DclXlrOptLength.
       MOVE 12 TO LstOptLength.
       MOVE "LIST FLAG(I)" to LstOpt.
       MOVE 12 to LstFilNamLength.
       MOVE "SAMPLE_G.LST" TO LstFilNam.
       CALL "FMTGEN" USING
                            BY REFERENCE DclXlrId
                            BY VALUE     ParameterCCSID
                            BY VALUE     DclXlrOptLength
                            BY REFERENCE DclXlrOpt
                            BY REFERENCE AdlDclSpc
                            BY VALUE AdlSpcCCSID
                            BY VALUE SrcFilNamLength
                            BY REFERENCE SrcFilNam
                            BY VALUE     SrcFilCCSID
                            BY VALUE     LstOptLength
                            BY REFERENCE LstOpt
                            BY VALUE     LstFilNamLength
                            BY REFERENCE LstFilNam
                            BY VALUE     LstFilCCSID
                            BY REFERENCE FeedBack.
****************************************************************
* Check the Condition Token                                    *
****************************************************************
       IF MsgNo OF Condition-ID IN FeedBack NOT = GEN-NO-ERROR
       THEN
            DISPLAY "Error in GENERATE function"
            DISPLAY
             "The Condition Token has the following contents:"
            DISPLAY "Message Severity "
                    MsgSev OF Condition-ID IN FeedBack
                    " Number "
                    MsgNo OF Condition-ID IN FeedBack
```

Figure 25 (Part 5 of 7). COBOL ADL Declaration Translator and CPB Sample

```
            DISPLAY "Case+Severity+Control "
                    Case-Severity-Control IN FeedBack
            DISPLAY "Facility ID      "
                    Facility-ID OF FeedBack
            DISPLAY "Instance Specific "
                    AdlExId OF FeedBack
            STOP RUN
       END-IF.
****************************************************************
* Call Conversion Plan Builder                                 *
* Note: If UserAdlPlnSpcCount (third parameter) is not zero,   *
*       then the fourth parameter must be replaced with        *
*            BY REFERENCE UserAdlPlnSpcList                    *
*       and addresses of user defined plan spaces must be      *
*       entered into the list.                                 *
****************************************************************
       MOVE 1 TO AdlDclSpcCount.
       SET AdlDclSpcPtr(1) TO ADDRESS OF AdlDclSpc.
       MOVE 0 TO UserAdlPlnSpcCount.
       MOVE 1 TO DefaultAdlPlnSpcCount.
       SET DefaultAdlPlnSpcPtr(1) TO ADDRESS OF AdlPlnSpc.
       MOVE BUFLEN-CNVPLNSPC TO CnvPlnSpcLength.
       MOVE 0 TO FlagList.
       SET AdlCommAreaPtr TO ADDRESS OF AdlCommArea.
       CALL "FMTCRCP" USING
                            BY VALUE     AdlDclSpcCount
                            BY REFERENCE AdlDclSpcList
                            BY VALUE     UserAdlPlnSpcCount
                            BY VALUE     UserAdlPlnSpcNull
                            BY VALUE     DefaultAdlPlnSpcCount
                            BY REFERENCE DefaultAdlPlnSpcList
                            BY VALUE     CnvPlnSpcLength
                            BY REFERENCE CnvPlnSpc
                            BY VALUE     FlagList
                            BY REFERENCE FeedBack.
****************************************************************
* Check the Condition Token                                    *
* Note: The Case-Severity-Control field is further divided into*
*       three sub-fields. You may want to display it in hex.   *
****************************************************************
       IF MsgNo OF Condition-ID IN FeedBack NOT = CPB-NO-ERROR
       THEN
            DISPLAY "Error in Conversion Plan Builder"
            DISPLAY
             "The Condition Token has the following contents:"
            DISPLAY "Message Severity "
                    MsgSev OF Condition-ID IN FeedBack
                    " Number "
                    MsgNo OF Condition-ID IN FeedBack
```

Figure 25 (Part 6 of 7). COBOL ADL Declaration Translator and CPB Sample

```
            DISPLAY "Case+Severity+Control "
                    Case-Severity-Control IN FeedBack
            DISPLAY "Facility ID "
                    Facility-ID OF FeedBack
****************************************************************
* Check whether an ADL exception occurred. If so, the ADL      *
* communication area is displayed.                             *
****************************************************************
            IF MsgNo OF Condition-ID = CPB-ADL-EXCEPTION-SEV2  OR
               MsgNo OF Condition-ID = CPB-ADL-EXCEPTION-SEV3
           THEN
               DISPLAY
                "The ADL communication area has the following
-                            " contents:"
               DISPLAY "ADL exception: " ExId OF AdlCommArea
               DISPLAY "Severity of ADL exception: "
                       SevCod OF AdlCommArea
               DISPLAY "Name of processed plan: "
                       CharData OF PlanId OF AdlCommArea
               DISPLAY "Number of processed PLAN statement: "
                       PlanStmt OF AdlCommArea
               DISPLAY "Source identifier of processed assignment
-                             " statement: "
                       CharData OF SrcFldId OF AdlCommArea
               DISPLAY "Target identifier of processed assignment
-                             " statement: "
                       CharData OF TrgFldId OF AdlCommArea
               STOP RUN
            END-IF
       ELSE
****************************************************************
* Write conversion plan space into file.                       *
****************************************************************
          OPEN OUTPUT Sample-SPC
          WRITE  CnvPlnSpc
          CLOSE SAMPLE-SPC.
       STOP RUN.
```

Figure 25 (Part 7 of 7). COBOL ADL Declaration Translator and CPB Sample

Conversion Plan Executor Example

```
****************************************************************
*                                                              *
* PRODUCT = Data Description and Conversion                    *
*                                                              *
* SOURCE FILE NAME = Sample2.CBL                               *
*                                                              *
* DESCRIPTIVE NAME = Conversion Plan Executor Example          *
*                                                              *
* FUNCTION = This sample program calls the function of the    *
*           conversion plan executor to convert data based on  *
*           the conversion plans created by the conversion     *
*           plan builder in program SAMPLE1.                   *
*           In this sample the hex string C1C2C301234C will    *
*           be converted with specified plan COBOL_TO_C into   *
*           the hex string 41424300D204. Then the hex string   *
*           41424300d204 will be converted with the specified  *
*           plan C_TO_COBOL into the hex string C1C2C301234C.  *
*           The result will be printed on screen.              *
*                                                              *
****************************************************************
  IDENTIFICATION DIVISION.
  PROGRAM-ID. SAMPLE2.
  ENVIRONMENT DIVISION.
  CONFIGURATION SECTION.
  SOURCE-COMPUTER. IBM-PS2.
  OBJECT-COMPUTER. IBM-PS2.
  INPUT-OUTPUT SECTION.
  FILE-CONTROL.
      SELECT Sample-SPC ASSIGN TO SampleF
                        ORGANIZATION IS SEQUENTIAL.
  DATA DIVISION.
  FILE SECTION.
  FD  Sample-SPC
           RECORD CONTAINS 5000 CHARACTERS
           RECORDING MODE IS F.
  01  CnvPlnSpc  PICTURE X(5000).
  WORKING-STORAGE SECTION.
```

Figure 26 (Part 1 of 8). COBOL Conversion Plan Executor Example

```
****************************************************************
* Parameters for FMTCPXI, FMTCPXC, and FMTCPXT                 *
****************************************************************
  77  CnvPlnSpcHdl PICTURE 9(6) USAGE IS BINARY.
  77  PlnNamLength PICTURE 9(6) USAGE IS BINARY.
  77  PlnNam PICTURE X(255).
  77  InputParmNum PICTURE 9(6) USAGE IS BINARY.
  77  OutputParmNum PICTURE 9(6) USAGE IS BINARY.
  01  InputData.
       05  InputDataPtr USAGE IS POINTER OCCURS 32 TIMES.
  01  OutputData.
       05  OutputDataPtr USAGE IS POINTER OCCURS 32 TIMES.
  01     FeedBack.
       05 Condition-ID.
          10 MsgSev  PICTURE 9(2) USAGE IS BINARY.
          10 MsgNo    PICTURE 9(2) USAGE IS BINARY.
       05 Case-Severity-Control  PICTURE X.
       05 Facility-ID  PICTURE X(3).
       05 I-S-Info.
          10 AdlExId  PICTURE 9(6) USAGE IS BINARY.
          10 AdlCommAreaPtr REDEFINES AdlExId USAGE IS POINTER.
          10 User-ExitCtokPtr  REDEFINES AdlExId USAGE IS POINTER.
****************************************************************
* ADL Communication Area                                       *
****************************************************************
  01     AdlCommArea.
      05 AdlCALength  PICTURE 9(6) USAGE IS BINARY.
      05 ExId  PICTURE 9(6) USAGE IS BINARY.
      05 SevCod  PICTURE 9(2) USAGE IS BINARY.
      05 PlanId.
          10  PreLength  PICTURE 9(2) USAGE IS BINARY.
          10  CharData  PICTURE X(255).
      05 PlanStmt  PICTURE 9(6) USAGE IS BINARY.
      05 InpErrDta.
          10  PreLength  PICTURE 9(2) USAGE IS BINARY.
          10  CharData  PICTURE X(255).
      05 SrcFldID.
          10  PreLength  PICTURE 9(2) USAGE IS BINARY.
          10  CharData  PICTURE X(255).
      05 TrgFldId.
          10  PreLength  PICTURE 9(2) USAGE IS BINARY.
          10  CharData  PICTURE X(255).
****************************************************************
* Return Codes and ADL Exception Codes                         *
****************************************************************
  77  CPX-NO-ERROR  PICTURE 9(6) USAGE IS BINARY
                    VALUE IS 0.
  77  CPX-ADL-EXCEPTION-SEV2  PICTURE 9(6) USAGE IS BINARY
                    VALUE IS 102.
```

Figure 26 (Part 2 of 8). COBOL Conversion Plan Executor Example

```
  77  CPX-ADL-EXCEPTION-SEV3  PICTURE 9(6) USAGE IS BINARY
                    VALUE IS 103.
  77  CpxType  PICTURE X(7).
****************************************************************
* Input and Output buffers and test data for SAMPLE2          *
****************************************************************
  77  SizeOfInput  PICTURE 9(6) USAGE IS BINARY VALUE IS 6.
  77  InValue  PICTURE X(6).
  01  OutBuffer.
       05 OutValue  PICTURE X(6).
       05 OutValueTbl REDEFINES OutValue
                      PICTURE X(1) OCCURS 6 TIMES
                      INDEXED BY M.
  01  OutHexBuffer VALUE IS "X'".
       05 OutHexValue PICTURE X(1) OCCURS 15 TIMES
                      INDEXED BY K.
  77  EBCD  PICTURE X(6)  VALUE IS X"C1C2C301234C".
  77  ASCII  PICTURE X(6)  VALUE IS X"41424300D204".
****************************************************************
* Tables and Work Areas for displaying the value in the output *
* buffer in hex                                                *
****************************************************************
  77  First-Digit  PICTURE 9(6) USAGE IS BINARY.
  77  Second-Digit  PICTURE 9(6) USAGE IS BINARY.
  77  I  PICTURE 9(6) USAGE IS BINARY.
  77  Switch  PICTURE 9(6) USAGE IS BINARY.
  01  HexIndex.
       05 HexValue-1  PICTURE X(128)
                       VALUE IS X"000102030405060708090A0B0C0D0E0F
-                                 "101112131415161718191A1B1C1D1E1F
-                                 "202122232425262728292A2B2C2D2E2F
-                                 "303132333435363738393A3B3C3D3E3F
-                                 "404142434445464748494A4B4C4D4E4F
-                                 "505152535455565758595A5B5C5D5E5F
-                                 "606162636465666768696A6B6C6D6E6F
-                               "707172737475767778797A7B7C7D7E7F".
      05 HexTable-1  REDEFINES HexValue-1
                         PICTURE X(1) OCCURS 128 TIMES.
       05 HexValue-2  PICTURE X(128)
                       VALUE IS X"808182838485868788898A8B8C8D8E8F
-                                 "909192939495969798999A9B9C9D9E9F
-                                 "A0A1A2A3A4A5A6A7A8A9AAABACADAEAF
-                                 "B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF
-                                 "C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF
-                                 "D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF
-                                 "E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF
-                             "F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF".
      05 HexTable-2  REDEFINES HexValue-2
                         PICTURE X(1) OCCURS 128 TIMES.
```

Figure 26 (Part 3 of 8). COBOL Conversion Plan Executor Example

```
  01  Hex-To-Char.
       05 CharValue  PICTURE X(16)
                     VALUE IS "0123456789ABCDEF".
       05 Hex-To-Char-Tbl  REDEFINES CharValue PICTURE X(1)
                           OCCURS 16 TIMES.
  PROCEDURE DIVISION.
****************************************************************
* Read the conversion plan space created in SAMPLE1            *
****************************************************************
      OPEN INPUT Sample-SPC.
      READ Sample-SPC.
      CLOSE Sample-SPC.
****************************************************************
* Call Conversion Plan Executor Initialization                 *
****************************************************************
      CALL "FMTCPXI" USING
                  BY REFERENCE CnvPlnSpc
                  BY REFERENCE CnvPlnSpcHdl
                  BY REFERENCE FeedBack.
****************************************************************
* Check the Condition Token                                    *
****************************************************************
      IF MsgNo OF Condition-ID IN FeedBack NOT = CPX-NO-ERROR
      THEN
         MOVE "Init" TO CpxType
         PERFORM PrintCtok
      ELSE
         SET AdlCommAreaPtr TO ADDRESS OF AdlCommArea
****************************************************************
* Call Conversion Plan Executor Convert                        *
* In this conversion the plan COBOL_TO_C is executed. The input*
* data are:                                                    *
*    "ABC"  in international EBCDIC format -> X'C1C2C3'        *
*    1234   in PACKED PRECISION(5) format  -> X'01234C'        *
*                                                              *
* The output data after conversion should be:                  *
*    "ABC"  in Latin PC Data format + suffix -> X'41424300'    *
*    1234   in BINARY byte reversed format   -> X'D204'        *
****************************************************************
         MOVE 10 TO PlnNamLength
         MOVE "COBOL_TO_C" TO PlnNam
         MOVE 1 TO InputParmNum
         MOVE 1 TO OutputParmNum
         MOVE EBCD TO InValue
         MOVE SPACES TO OutValue
         SET InputDataPtr(1) TO ADDRESS OF InValue
         SET OutputDataPtr(1) TO ADDRESS OF OutValue
```

Figure 26 (Part 4 of 8). COBOL Conversion Plan Executor Example

```
         CALL "FMTCPXC" USING
                     BY VALUE     CnvPlnSpcHdl
                     BY VALUE     PlnNamLength
                     BY REFERENCE PlnNam
                     BY VALUE     InputParmNum
                     BY REFERENCE InputData
                     BY VALUE     OutputParmNum
                     BY REFERENCE OutputData
                     BY REFERENCE FeedBack
         IF MsgNo OF Condition-ID IN FeedBack NOT = CPX-NO-ERROR
         THEN
****************************************************************
* An error occurred. Print the condition token                 *
****************************************************************
            MOVE "Convert" TO CpxType
            PERFORM PrintCtok
            STOP RUN
         ELSE
****************************************************************
* Print the converted value                                    *
****************************************************************
            SET K TO 3
            PERFORM Hex-Convert VARYING M FROM 1 BY 1
                                UNTIL M > SizeOfInput
            MOVE "'" TO OutHexValue(K)
            DISPLAY " Converted value for plan COBOL_TO_C: "
                                          OutHexBuffer
         END-IF
****************************************************************
* Call Conversion Plan Executor Convert                        *
* In this conversion the plan C_TO_COBOL is executed. The input*
* data are:                                                    *
*    "ABC"  in Latin PC Data format + suffix -> X'41424300'    *
*    1234   in BINARY byte reversed format   -> X'D204'        *
*                                                              *
* The output data after conversion should be:                  *
*    "ABC"  in international EBCDIC format -> X'C1C2C3'        *
*    1234   in PACKED PRECISION(5) format  -> X'01234C'        *
****************************************************************
         MOVE 10 TO PlnNamLength
         MOVE "C_TO_COBOL" TO PlnNam
         MOVE 1 TO InputParmNum
         MOVE 1 TO OutputParmNum
         MOVE ASCII TO InValue
         MOVE SPACES TO OutValue
         SET InputDataPtr(1) TO ADDRESS OF InValue
         SET OutputDataPtr(1) TO ADDRESS OF OutValue
```

Figure 26 (Part 5 of 8). COBOL Conversion Plan Executor Example

```
         CALL "FMTCPXC" USING
                     BY VALUE     CnvPlnSpcHdl
                     BY VALUE     PlnNamLength
                     BY REFERENCE PlnNam
                     BY VALUE     InputParmNum
                     BY REFERENCE InputData
                     BY VALUE     OutputParmNum
                     BY REFERENCE OutputData
                     BY REFERENCE FeedBack
         IF MsgNo OF Condition-ID IN FeedBack NOT = CPX-NO-ERROR
         THEN
****************************************************************
* An error occurred. Print the condition token                 *
****************************************************************
            MOVE "Convert" TO CpxType
            PERFORM PrintCtok
         ELSE
****************************************************************
* Print the converted value                                    *
****************************************************************
            SET K TO 3
            PERFORM Hex-Convert VARYING M FROM 1 BY 1
                                UNTIL M > SizeOfInput
            MOVE "'" TO OutHexValue(K)
            DISPLAY " Converted value for plan C_TO_COBOL: "
                                                 OutHexBuffer
         END-IF
****************************************************************
* Call Conversion Plan Executor Termination                    *
****************************************************************
         CALL "FMTCPXT" USING
                     BY VALUE     CnvPlnSpcHdl
                     BY REFERENCE FeedBack
****************************************************************
* Check the Condition Token                                    *
****************************************************************
         IF MsgNo OF Condition-ID IN FeedBack NOT = CPX-NO-ERROR
         THEN
            MOVE "Term" TO CpxType
            PERFORM PrintCtok
         END-IF
      END-IF.
      STOP RUN.
```

Figure 26 (Part 6 of 8). COBOL Conversion Plan Executor Example

```
****************************************************************
* The procedure PrintCtok prints the condition token and ADL   *
* communication area after an error occurred in a conversion   *
* plan executor function.                                      *
* Note: The Case-Severity-Control field is further divided into*
*       three sub-fields. You may want to display it in hex.   *
****************************************************************
   PrintCtok.
      DISPLAY "Error in Conversion Plan Executor " CpxType.
      DISPLAY "The Condition Token has the following contents:".
      DISPLAY "Message Severity "
              MsgSev OF Condition-ID IN FeedBack
              " Number "
              MsgNo OF Condition-ID IN FeedBack.
      DISPLAY "Case+Severity+Control "
              Case-Severity-Control IN FeedBack.
      DISPLAY "Facility ID "
                    Facility-ID OF FeedBack.
****************************************************************
* Check whether an ADL exception occurred.                     *
****************************************************************
      IF MsgNo OF Condition-ID = CPX-ADL-EXCEPTION-SEV2  OR
          MsgNo OF Condition-ID = CPX-ADL-EXCEPTION-SEV3
      THEN
         IF CpxType = "INIT"
         THEN
            DISPLAY "ADL exception " AdlExId IN FeedBack
         ELSE
            DISPLAY
             "The ADL communication area has the following
-                         " contents:"
            DISPLAY "ADL exception: " ExId OF AdlCommArea
            DISPLAY "Severity of ADL exception: "
                    SevCod OF AdlCommArea
            DISPLAY "Name of processed plan: "
                    CharData OF PlanId OF AdlCommArea
            DISPLAY "Number of processed PLAN statement: "
                    PlanStmt OF AdlCommArea
            DISPLAY "Input data portion that caused the error: "
                    CharData OF InpErrDta OF AdlCommArea
            DISPLAY "Source identifier of processed assignment
-                          " statement: "
                    CharData OF SrcFldId OF AdlCommArea
            DISPLAY "Target identifier of processed assignment
-                          " statement: "
                    CharData OF TrgFldId OF AdlCommArea
         END-IF
      END-IF.
```

Figure 26 (Part 7 of 8). COBOL Conversion Plan Executor Example

```
****************************************************************
* The procedure Hex-Convert converts a byte string to a hex    *
* string which can be DISPLAYed.                               *
****************************************************************
   Hex-Convert.
      MOVE 0 TO Switch.
      PERFORM Hex-Convert-1 VARYING I FROM 1 BY 1
                                  UNTIL Switch = 1 OR I > 128.
      IF Switch = 0
      THEN
         PERFORM Hex-Convert-2 VARYING I FROM 1 BY 1
                                  UNTIL Switch = 1
      END-IF.
   Hex-Convert-1.
      IF OutValueTbl(M) = HexTable-1(I)
      THEN
         DIVIDE 16 INTO I GIVING First-Digit
                          REMAINDER Second-Digit
         MOVE Hex-To-Char-Tbl(First-Digit + 1)
                              TO OutHexValue(K)
         SET K UP BY 1
         MOVE Hex-To-Char-Tbl(Second-Digit) TO OutHexValue(K)
         SET K UP BY 1
         MOVE 1 TO Switch
      END-IF.
   Hex-Convert-2.
      IF OutValueTbl(M) = HexTable-2(I)
      THEN
         DIVIDE 16 INTO I GIVING First-Digit
                          REMAINDER Second-Digit
         MOVE Hex-To-Char-Tbl(First-Digit + 9)
                     TO OutHexValue(K)
         SET K UP BY 1
         MOVE Hex-To-Char-Tbl(Second-Digit) TO OutHexValue(K)
         SET K UP BY 1
         MOVE 1 TO Switch
      END-IF.
```

Figure 26 (Part 8 of 8). COBOL Conversion Plan Executor Example

User Exit (ADL Source) Example

```
/*************************************************************/
/*  USEREXIT.ADL                                              */
/*  ADL source for the USER EXIT sample.                      */
/*************************************************************/

/*-----------------DECLARE of the source data----------------*/
source: DECLARE
 BEGIN;
    number_of_char: CONSTANT 10;
    input1:         CHAR LENGTH( number_of_char);
 END;
/*-----------------------------------------------------------*/

/*-----------------DECLARE OF the target data----------------*/
target: DECLARE
 BEGIN;
    output1:        CHAR LENGTH( 20 );
    output2:        CHAR LENGTH( 20 );
 END;
/*-----------------------------------------------------------*/
/*-----------------PLAN--------------------------------------*/
user_exit_plan: PLAN ( input1: INPUT,
                       output1: OUTPUT,
                       output2: OUTPUT )
 BEGIN;
    CALL '<USEREXIT><CONVERTTOUPPERLOWERCASECHAR>'
                   ( input1,
                     LENGTH( input1 ),
                     TRUE,  /* Convert to upper case character */
                     output1,
                     LENGTH( output1 )
                   );
    CALL '<USEREXIT><CONVERTTOUPPERLOWERCASECHAR>'
                   ( input1,
                     LENGTH( input1 ),
                     FALSE, /* Convert to lower case character */
                     output2,
                     LENGTH( output2 )
                   );
 END;
/*-----------------------------------------------------------*/
```

Figure 27. User Exit (ADL Source) Example

Sample ADL File

```
DECLARE BEGIN;         /* ADL */
  COBOLREC: SEQUENCE BEGIN;
    INITIALS: CHAR LENGTH(3)
              CCSID(500);
    NUMBER: PACKED PRECISION(5);
  END;
END;

DECLARE BEGIN;         /* ADL */
  CREC: SEQUENCE BEGIN;
    INITIALS: CHARSFX MAXLEN(4)
              CCSID(850);
    NUMBER: BINARY PRECISION(15)
            BYTRVS (TRUE );
  END;
END;

COBOL_TO_C: PLAN( COBOLREC: INPUT,
                  CREC: OUTPUT )
  BEGIN;
    CREC <- COBOLREC;
  END;

C_TO_COBOL: PLAN( CREC: INPUT,
                  COBOLREC: OUTPUT )
  BEGIN;
    COBOLREC <- CREC;
  END;
```

Figure 28. Sample ADL File

Appendix C. Sample Programs in PLI

The following are sample programs in PLI.

TEST.PLI

```
/***************************************************************************/
/*                                                                         */
/* PRODUCT   = Data Description and Conversion for OS/2                    */
/*                                                                         */
/* SOURCE FILE NAME = TEST.PLI                                             */
/*                                                                         */
/* DESCRIPTIVE NAME = ADL Declaration Translator, CPB, and CPEX sample     */
/*                                                                         */
/* FUNCTION =  This TEST.PLI program parses the adl file TEST.ADL using the*/
/*             FMTPRS function and return the appropriate ADL declare and  */
/*             plan space.  It then invokes FMTCRCP to create conversion   */
/*             plan from the encoded ADL declare and plan space.  With the */
/*             conversion plan, TEST.PLI invokes the FMTCPXI to initialize */
/*             conversion plan executor, calls FMTCPXC to executes the     */
/*             conversion plan CNVREC to convert EBCDIC string to ASCII    */
/*             string, and finally calls FMTCPXT to terminate and release  */
/*             resources.                                                  */
/*             In summary, TEST.PLI exercises FMTPRS, FMTCRCP, FMTCPXI,    */
/*             FMTCPXC, and FMTCPXT functions to accomplish the task of    */
/*             convert EBCDIC string to ASCII string as specified by the   */
/*             TEST.ADL.                                                   */
/*                                                                         */
/* NOTES =                                                                 */
/*                                                                         */
/*   DEPENDENCIES = OS/2 Release 2.0 or later                              */
/*                                                                         */
/*   RESTRICTIONS = None                                                   */
/*                                                                         */
/* ENTRY POINTS = test()                                                   */
/*                                                                         */
/*                                                                         */
/***************************************************************************/

/*************************************/
/* Program variables                 */
/*************************************/
DCL ADLDECL  CHAR(9) VARZ;
DCL CCSID1 FIXED BIN(31,0);
DCL ADLFILELEN FIXED BIN(31,0);
DCL ADLFILENAM CHAR(9) VARZ;
DCL CCSID2 FIXED BIN(31,0);
DCL XLROPTLEN FIXED BIN(31,0);
DCL XLROPTNAME CHAR(9) VARZ;
DCL LSTOPTLEN FIXED BIN(31,0);
DCL LSTOPTNAME CHAR(9) VARZ;
DCL LSTLEN FIXED BIN(31,0);
DCL LSTNAME CHAR(9) VARZ;
DCL ADLDCLLEN FIXED BIN(31,0);
DCL ADLDCLSPC CHAR(8000) CONTROLLED;
DCL CCSID3 FIXED BIN(31,0);
```

Figure 29 (Part 1 of 8). TEST.PLI

```
/***************************************/
/* Mapping CNSTKN and FMTCTOK in fmt.h */
/***************************************/

DCL 1 CNSTKN UNALIGNED,
     2 ULLENGTH FIXED BIN(31,0),
     2 USCLASS  FIXED BIN(15,0),
     2 ABVALUE  CHAR(16) VARZ;
DCL PCNSTKN POINTER;
DCL ADLPLNLEN FIXED BIN(31,0);
DCL ADLPLNSPC CHAR(8000) CONTROLLED;

DCL 1 FMTADLCA UNALIGNED,
     2 LLENGTH FIXED BIN(31,0),
     2 LEXID   FIXED BIN(31,0),
     2 USSEVCOD FIXED BIN(15,0),
     2 PLANID,
       3 USLENGTH FIXED BIN(15,0),
       3 UCHDATA CHAR(255),
     2 LPLANSTMT FIXED BIN(31,0),
     2 INPERRDTA,
       3 USLENGTH FIXED BIN(15,0),
       3 UCHDATA CHAR(255),
     2 SRCFLDID,
       3 USLENGTH FIXED BIN(15,0),
       3 UCHDATA CHAR(255),
     2 TRGFLDID,
       3 USLENGTH FIXED BIN(15,0),
       3 UCHDATA CHAR(255);

DCL 1 FMTCTOK UNALIGNED,
     2 CONDITION_ID,
       3 USMSGSEV FIXED BIN(15,0),
       3 USMSGNO  FIXED BIN(15,0),
     2 FCASE BIT(2),
     2 FSEVERITY BIT(3),
     2 FCONTROL BIT(3),
     2 UCHFACILITY_ID CHAR(3),
     2 PL_S_INFO UNION,
       3 ULADLEXLD FIXED BIN(31,0),
       3 PADLCOMMAREA POINTER,
       3 PUSEREXITCTOK POINTER ;

DCL PFMTCTOK POINTER;
```

Figure 29 (Part 2 of 8). TEST.PLI

```
/***************************************/
/* Program variables                   */
/***************************************/

DCL 1 X ,
     2 C CHAR(5);

DCL 1 Y ,
     2 C CHAR(10);

DCL 1 PPADLDCLSPCLIST POINTER;
DCL 1 PPDEFAULTADLPLNSPCLIST POINTER;
DCL 1 PPUSERADLPLNSPCLIST POINTER;
DCL 1 CNVPLNSPC CHAR(8000) CONTROLLED;

DCL 1 ULCNVPLNSPCHDL FIXED BIN(31,0);
DCL 1 PULCNVPLNSPCHDL POINTER;

DCL 1 PINPUTDATA POINTER;
DCL 1 PPINPUTDATA POINTER;
DCL 1 POUTPUTDATA POINTER;
DCL 1 PPOUTPUTDATA POINTER;

/***************************************/
/* Prototype for DDC API               */
/***************************************/

DCL FMTPRS entry( char(*) varz byaddr,fixed bin(31) byvalue,
                    fixed bin(31) byvalue,char(*) varz byaddr,
                    fixed bin(31) byvalue,
                    fixed bin(31) byvalue,char(*) varz byaddr,
                    fixed bin(31) byvalue,char(*) varz byaddr,
                    fixed bin(31) byvalue,char(*) varz byaddr,
                    fixed bin(31) byvalue,char(*) byaddr,
                    fixed bin(31) byvalue,
                    pointer byvalue,
                    fixed bin(31) byvalue,char(*) byaddr,
                    pointer byvalue)
                    external('FMTPRS')
                    options(linkage(system));
```

Figure 29 (Part 3 of 8). TEST.PLI

```
  DCL FMTCRCP entry( fixed bin(31) byvalue,
                     pointer byaddr,
                     fixed bin(31) byvalue,
                     pointer byvalue,
                     fixed bin(31) byvalue,
                     pointer byaddr,
                     fixed bin(31) byvalue,
                     char(*) byaddr,
                     fixed bin(31) byvalue,
                     pointer byvalue)
                     external('FMTCRCP')
                     options(linkage(system));

  DCL FMTCPXI entry( char(*) byaddr,
                     pointer byvalue,
                     pointer byvalue)
                     external('FMTCPXI')
                     options(linkage(system));

  DCL FMTCPXC entry( fixed bin(31) byvalue,
                     fixed bin(31) byvalue,
                     char(*) byaddr,
                     fixed bin(31) byvalue,
                     pointer byvalue,
                     fixed bin(31) byvalue,
                     pointer byvalue,
                     pointer byvalue)
                     external('FMTCPXC')
                     options(linkage(system));

  DCL FMTCPXT entry( fixed bin(31) byvalue,
                     pointer byvalue)
                     external('FMTCPXT')
                     options(linkage(system));

  ALLOCATE ADLDCLSPC;
  ALLOCATE ADLPLNSPC;
```

Figure 29 (Part 4 of 8). TEST.PLI

```
ADLDECL = '2B12000301886D01'X;
CCSID1 = 0;
ADLFILELEN = 8;
ADLFILENAM = "test.adl";
CCSID2 = 0;
XLROPTLEN = 8;
XLROPTNAME = "AUTOSKIP";
LSTOPTLEN = 4;
LSTOPTNAME = "LIST";
LSTLEN = 8;
LSTNAME = "test.lsp";
ADLDCLLEN = 8000;
ADLDCLSPC = REPEAT('00'X,4000);
CCSID3 = 0;
CNSTKN.ULLENGTH = 0;
CNSTKN.USCLASS  = 0;
CNSTKN.ABVALUE  = "THIS IS A TEST";
PCNSTKN = ADDR(CNSTKN);
ADLPLNLEN = 8000;
ADLPLNSPC = REPEAT('00'X,4000);
PFMTCTOK = ADDR(FMTCTOK);

fetch FMTPRS title('FMTB/FMTPRS');
call FMTPRS(ADLDECL,
              CCSID1,
              ADLFILELEN,
              ADLFILENAM,
              CCSID2,
              XLROPTLEN,
              XLROPTNAME,
              LSTOPTLEN,
              LSTOPTNAME,
              LSTLEN,
              LSTNAME,
              ADLDCLLEN,
              ADLDCLSPC,
              CCSID3,
              PCNSTKN,
              ADLPLNLEN,
              ADLPLNSPC,
              PFMTCTOK);
if FMTCTOK.CONDITION_ID.USMSGNO [= 0 then
  do;
    display('error in parsing the ADL file.');
    display('The condition token has the following contents:');
    put edit ('Message Severity :',FMTCTOK.CONDITION_ID.USMSGSEV)
             (skip,a(40),f(7));
    put edit ('Message Number   :',FMTCTOK.CONDITION_ID.USMSGNO)
             (skip,a(40),f(7));
    goto exit;
  end;
```

Figure 29 (Part 5 of 8). TEST.PLI

```
else
  do;
    display('no error in parsing the ADL file.');
  end;

ALLOCATE CNVPLNSPC;
CNVPLNSPC = REPEAT('00'X,4000);
FMTCTOK.PL_S_INFO.PADLCOMMAREA = ADDR(FMTADLCA);

PPADLDCLSPCLIST = ADDR(ADLDCLSPC);
PPDEFAULTADLPLNSPCLIST = ADDR(ADLPLNSPC);
PPUSERADLPLNSPCLIST = NULL();

fetch FMTCRCP title('FMTC/FMTCRCP');
call FMTCRCP(1,
             PPADLDCLSPCLIST,
             0,
             NULL,
             1,
             PPDEFAULTADLPLNSPCLIST,
             8000,
             CNVPLNSPC,
             0,
             PFMTCTOK);
if FMTCTOK.CONDITION_ID.USMSGNO [= 0 then
  do;
    display('error in conversion plan builder.');
    display('The condition token has the following contents:');
    put edit ('Message Severity :',FMTCTOK.CONDITION_ID.USMSGSEV)
             (skip,a(40),f(7));
    put edit ('Message Number   :',FMTCTOK.CONDITION_ID.USMSGNO)
             (skip,a(40),f(7));
    goto exit;
  end;
else
  display('no error in conversion plan builder.');

PULCNVPLNSPCHDL = ADDR(ULCNVPLNSPCHDL);
fetch FMTCPXI title('FMTD/FMTCPXI');
call FMTCPXI(CNVPLNSPC,
             PULCNVPLNSPCHDL,
             PFMTCTOK);
if FMTCTOK.CONDITION_ID.USMSGNO [= 0 then
  do;
    display('error in initialize conversion plan executor.');
    display('The condition token has the following contents:');
    put edit ('Message Severity :',FMTCTOK.CONDITION_ID.USMSGSEV)
             (skip,a(40),f(7));
    put edit ('Message Number   :',FMTCTOK.CONDITION_ID.USMSGNO)
             (skip,a(40),f(7));
    goto exit;
```

Figure 29 (Part 6 of 8). TEST.PLI

```
  end;
else
  display('no error in initialize conversion plan executor.');

X.C = 'D3D4D5D6D7'X;

PINPUTDATA = ADDR(X);
PPINPUTDATA = ADDR(PINPUTDATA);
POUTPUTDATA = ADDR(Y);
PPOUTPUTDATA = ADDR(POUTPUTDATA);

fetch FMTCPXC title('FMTD/FMTCPXC');
call FMTCPXC(ULCNVPLNSPCHDL,
             6,
             "CNVREC",
             1,
             PPINPUTDATA,
             1,
             PPOUTPUTDATA,
             PFMTCTOK);
if FMTCTOK.CONDITION_ID.USMSGNO [= 0 then
  do;
    display('error in conversion plan executor convert.');
    display('The condition token has the following contents:');
    put edit ('Message Severity :',FMTCTOK.CONDITION_ID.USMSGSEV)
             (skip,a(40),f(7));
    put edit ('Message Number   :',FMTCTOK.CONDITION_ID.USMSGNO)
             (skip,a(40),f(7));
    goto exit;
  end;
else
  display('no error in conversion plan executor convert.');


fetch FMTCPXT title('FMTD/FMTCPXT');
call FMTCPXT(ULCNVPLNSPCHDL,
             PFMTCTOK);
if FMTCTOK.CONDITION_ID.USMSGNO [= 0 then
  do;
    display('error in terminate conversion plan executor.');
    display('The condition token has the following contents:');
    put edit ('Message Severity :',FMTCTOK.CONDITION_ID.USMSGSEV)
             (skip,a(40),f(7));
    put edit ('Message Number   :',FMTCTOK.CONDITION_ID.USMSGNO)
             (skip,a(40),f(7));
    goto exit;
  end;
```

Figure 29 (Part 7 of 8). TEST.PLI

```
  else
    do;
      display('no error in terminate conversion plan executor.');
      put edit ('Expected output for Y.C =               LMNOP')
               (skip,a(45));
      put edit ('Y.C = ',Y.C)
               (skip,a(40),a);
    end;

  exit:

  FREE ADLDCLSPC;
  FREE ADLPLNSPC;
  FREE CNVPLNSPC;

end test;
```

Figure 29 (Part 8 of 8). TEST.PLI

TEST.ADL

```
/********************************************/
/*          TEST ADL                        */
/********************************************/
DECLARE BEGIN;
 X: SEQUENCE BEGIN;
    C: CHAR LENGTH(5) CCSID(500);
 END;
END;

DECLARE BEGIN;
 Y: SEQUENCE BEGIN;
    C: CHAR LENGTH(10) CCSID(437);
 END;
END;

CNVREC: PLAN (X: INPUT,
              Y: OUTPUT)
  BEGIN;
    Y <- X;
  END;
```

Figure 30. TEST.ADL

TEST.MAK

```
/***************************************************/
/*                     TEST MAK                    */
/***************************************************/

doit: test.exe

test.exe: test.obj
        link386 test.obj /stack:32000 /co /noe,test.exe,,d:\pli\lib\ibmlink.lib d:\pli\lib\ceelink,test.def

test.obj: test.pli test.mak
        pli test.pli
```

Figure 31. TEST.MAK

TEST.DEF

```
/***************************************************/
/*                     TEST DEF                    */
/***************************************************/
=========================================================================
NAME TEST WINDOWCOMPAT
=========================================================================
```

Figure 32. TEST.DEF

Appendix D. Using the OS/2 Trace Function

The trace function is primarily used by IBM service personnel to debug internal problems that occur while using DD&C for OS/2. This trace function is not available on the AIX or Windows platforms.

The procedure for tracing is:

1. Prepare your system for tracing and define the level of trace events you want to collect.
2. Start the trace function.
3. Run the application program causing the error.
4. Stop the trace function.
5. Specify the ASCII file where you want the trace entries written.

Preparing to Use the Trace Function

Before you can use the DD&C trace function, you must make the following changes to your `CONFIG.SYS` file:

1. Set the path to the DD&C for OS/2 program files directory.

   To set the path to the DD&C program files directory to an environment variable:

   - Edit your CONFIG.SYS file, using an editor such as the OS/2 system editor. Add the following entry, and save the change:

     `SET FMTDIR=` *`program files directory`*

     Where *program files directory* is the directory where the DD&C for OS/2 program files are located. The default is `\IBMDDC`.

2. Specify the trace level. By specifying a trace level you define which objects or processes are to be traced. This way you receive only trace entries that are of relevant to your problem.

   If you do not specify a trace level, all events in DD&C for OS/2 are traced. (This corresponds to FMTTRACELEVEL=15 as shown in Table 10.)

   To specify a trace level:

   - Edit your `CONFIG.SYS` file, using an editor such as the OS/2 system editor. Add the following entry, and save the change:

     ```
     SET FMTTRACELEVEL = n
     ```

     Where `n` is a value between 1 and 15 inclusive. The meaning of the values is explained in Table 10 on page 222.

3. Activate the FMTTRACELEVEL environment variable. One way to do this is by shutting down and restarting your system.

Table 10 explains the trace levels available and the use of parameters for each level. The parameters are described as follows:

FMTTRACELEVEL A value between 1 and 15 with which you control the tracing of the objects and processes listed in the table.

API Parameter If `ON`, all parameter values or contents are traced.

Data If `ON`, all data buffers before the conversion and after the conversion are traced.

Locking If `ON`, each locking and unlocking of internal resources is recorded by two trace entries:

- One trace entry before locking or unlocking
- One trace entry after locking or unlocking, plus the return code of the lock or unlock command.

Internal Return Code If set to `ON`, the return codes of certain internal functions are traced.

Table 10. Trace levels available for FMTTRACE

FMTTRACELEVEL	API Parameter	Data	Locking	Internal Return Code
1	ON	OFF	OFF	OFF
2	OFF	ON	OFF	OFF
3	ON	ON	OFF	OFF
4	OFF	OFF	ON	OFF
5	ON	OFF	ON	OFF
6	OFF	ON	ON	OFF
7	ON	ON	ON	OFF
8	OFF	OFF	OFF	ON
9	ON	OFF	OFF	ON
10	OFF	ON	OFF	ON
11	ON	ON	OFF	ON
12	OFF	OFF	ON	ON
13	ON	OFF	ON	ON
14	OFF	ON	ON	ON
15	ON	ON	ON	ON

Issue the FMTTRACE Command

You issue the FMTTRACE command from an OS/2 window or full-screen command prompt. The syntax of the FMTTRACE command is:

```
FMTTRACE [
              ON [ [{/|-}B tracebuffersize]] |
              [ OFF ]
              [ {/|-}P [trace_file_name]
                     [ {/|-}N max_trace_entries] ]
              {/|-}HELP |
              {/|-}Q |
              {/|-}KILL
              ]
```

The parameters have the following meaning:

Parameter **Meaning**

ON Start collecting trace entries for DD&C API commands.

B Specifies the maximum amount of storage to be allocated for the trace entries. *tracebuffersize* is specified in bytes. If you do not specify *tracebuffersize*, a default of 64KB is used.

OFF Stop collecting trace entries.

P Writes collected trace entries. If the trace is still running, it is stopped when you issue this parameter. The entries are written to the *trace_file_name*. If you do not specify a *trace_file_name*, the entries are written to the screen.

N Specifies the maximum number of trace entries to be written. If you do not specify a value for *max_trace_entries*, all trace entries are either written to a trace file or to the screen.

HELP Displays help text explaining the use of the FMTTRACE command. If you enter the FMTTRACE command without any parameters, the help screen appears automatically.

Q Suppresses the title.

KILL Frees all resources allocated with the first FMTTRACE ON command.

Start the Trace Function

1. Go to an OS/2 window or full-screen command prompt.
2. At the prompt, enter:

   ```
   FMTTRACE ON
   ```

   This activates the trace function and the collected entries are stored.
3. Restart the application that causes the error.

Stop the Trace Function

1. Go to an OS/2 window or full-screen command prompt.
2. At the prompt, enter:

   ```
   FMTTRACE OFF
   ```

No further trace entries are generated until you restart FMTTRACE.

Write the Trace Entries to a File

Choose from the following options:

Viewing the trace entries on the screen

Issue the FMTTRACE /P command, but omit the *trace_file_name*.

Storing the entries in an ASCII file

When you start the trace, specify the name of a file to receive the trace entries. For example:

```
FMTTRACE /P trace.out
```

Specifying the number of trace entries to be displayed or stored.

To limit the number of trace entries to be displayed, specify the number when starting the trace. For example:

```
FMTTRACE /N 20
```

To limit the number of trace entries to be written to a file, specify the file name and the number when starting the trace.
For example:

```
FMTTRACE /P trace.out /N 20
```

Notes:

1. If you do not specify a number, all collected trace entries are either displayed or written to the file specified.
2. If you use an existing filename, FMTTRACE overwrites the existing information.
3. Printing the trace entries to a file implicitly stops FMTTRACE.

Example of Trace Output

Figure 33 shows the layout of the trace entries.

```
---------------------------- Begin of Trace Output  -------------------
 . . .
-----------------------------------------------------------------------
TRACELEVEL:  1 PID: 00419 TID: 0001 DATE: 07/25/1993 TIME: 16:02:59.47
TITLE: Input data to convert        LENGTH: 00023 MOD: FMTDCPXC ID: 01
00035C94: 6461 7461 2062 7566 6665 7220 746F 2063 | data buffer to c
00035CA4: 6F6E 7665 7274 00                       | onvert.
-----------------------------------------------------------------------
 . . .
-----------------------------------------------------------------------
TRACELEVEL:  1 PID: 00419 TID: 0001 DATE: 07/25/1993 TIME: 16:03:02.56???
TITLE: RC                           LENGTH: 00004 MOD: FMTDCPXC ID: 02
18883E36: 0000 0000                               | ....
----------------------------- End of Trace Output  --------------------
```

Figure 33. Layout of Trace Entries

The trace header for each entry consists of:

- The trace level of that trace entry
- The current OS/2 process identifier (PID)
- The current OS/2 thread identifier (TID)
- The date and time of the trace entry
- The title of the trace entry
- The length of the trace entry, as a decimal number
- The module name
- The ID of the trace entry point within a module.

Each line of a trace entry consists of:

- A pointer to the traced data area
- The trace entry data in hexadecimal format
- The readable trace entry data.

“Trace Function Messages” lists all error messages that can occur while using the DD&C trace facility.

Trace Function Messages

The following error messages can occur when using the DD&C for OS/2 trace function (FMTTRACE):

FMT0002E Error opening file *fn*.

Explanation: The program is unable to open the specified file. The file does not exist, cannot be created, or the drive containing the file is not ready.

User Response: Correct the problem and then start FMTTRACE again.

FMT0003E Parameter *prm* for the command is not valid.

Explanation: The parameter you have specified is not valid for this command.

User Response: Refer to "Issue the FMTTRACE Command" on page 222 for a list of valid keywords. Specify the command with the correct keyword.

FMT0004E Semaphores cannot be created.

Explanation: DD&C for OS/2 attempted to create an OS/2 semaphore in order to serialize access to shared resources, but creation failed due to a system error.

User Response: Contact your IBM support representative.

FMT0005E The detached process FMTGTDAE.EXE cannot be started.

Explanation: To keep the trace data resident in memory, DD&C for OS/2 needs a detached process, FMTGTDAE.EXE. This program cannot be found. DD&C for OS/2 stops processing.

User Response: Verify that the program FMTGTDAE.EXE is available in the DD&C for OS/2 program files directory and that the environment variable FMTDIR contains the name of this system directory.

FMT0006E The detached process FMTGTDAE.EXE, used to keep trace data, was started, but the central resources in the shared segment cannot be accessed.

Explanation: To keep the trace resources resident in memory, DD&C for OS/2 requires a detached process, FMTGTDAE.EXE. FMTGTDAE.EXE cannot access the shared memory where the resources are located. DD&C for OS/2 stops processing. This is a system error.

User Response: Contact your IBM support representative.

FMT0007E Internal semaphore error.

Explanation: An internal system error occurred when DD&C for OS/2 attempted to use an OS/2 semaphore to serialize access to shared resources.

User Response: Contact your IBM support representative.

FMT0008E HELP and Q parameters are mutually exclusive.

Explanation: The parameters HELP and Q (suppress title) cannot be specified at the same time when you invoke FMTTRACE.

User Response: Specify either HELP or Q, but not both.

FMT0009I FMTTRACE processing complete.

Explanation: The trace has been turned off.

User Response: None

FMT0011E ON and OFF parameters are mutually exclusive.

Explanation: The parameters ON and OFF cannot be specified at the same time when you invoke FMTTRACE.

User Response: Specify either ON or OFF, but not both.

FMT0012E N parameter was specified but FMTTRACE was not invoked with the P parameter.

Explanation: The N (maximum trace entries) parameter can only be specified together with the P (activate trace print) parameter.

User Response: Remove the N parameter or add the P parameter.

FMT0013E You tried to activate the DD&C for OS/2 trace and print the trace entries at the same time. Printing, however, implicitly ends the trace collection.

Explanation: The ON parameter starts the collection of trace entries and the P parameter prints the trace entries already collected. There are no entries to print when the P parameter is specified with the ON parameter.

User Response: Do not specify the P parameter with the ON parameter.

FMT0014I The DD&C for OS/2 trace is already active.

Explanation: You attempted to start the DD&C for OS/2 trace, but it has already been started and is still active.

User Response: To restart the trace, invoke the FMTTRACE OFF command first.

FMT0015I The DD&C for OS/2 trace is already inactive.

Explanation: You attempted to stop the DD&C for OS/2 trace, but it has already been stopped.

FMT0016E The DD&C for OS/2 trace memory is in use. The trace memory call timed out.

Explanation: A process attempted to access the trace memory, but a timeout problem occurred.

User Response: Try the command again. If the same message occurs, contact your IBM support representative.

FMT0017E The DD&C for OS/2 trace memory exists, but cannot be accessed. DosRequestMutexSem returned *rc*.

Explanation: The specified error occurred when DD&C for OS/2 attempted to use an OS/2 semaphore to access shared resources.

User Response: Contact your IBM support representative.

FMT0018E DD&C for OS/2 could not allocate the shared memory required for tracing.

Explanation: DD&C for OS/2 attempted to allocate shared memory for tracing that already exists.

User Response: Contact your IBM support representative.

FMT0019E Either -B or /B can only be used with ON at the same time when you invoke FMTTRACE.

Explanation: FMTTRACE was invoked with the B (buffer size) parameter, but without the ON parameter. If the parameter B is specified, the ON parameter must also be specified.

User Response: Only specify the B parameter together with the ON parameter.

FMT0020E The environment variable FMTDIR is not set.

Explanation: FMTTRACE was invoked but the environment variable FMTDIR has not previously been set.

User Response: Set the environment variable:

```
SET FMTDIR=x:\IBMDDC
```

Where *x* is the drive letter of the drive where DD&C for OS/2 is installed.

Glossary

A Data Language. A language for describing the fields, arrays, and so on of data records in a programming environment so the records can be transparently accessed by other programming environments.

abend. Abnormal end of task.

access method. The part of the DDM architecture which accepts commands to access and process the records of a file.

ADL. A Data Language

ADLCA. ADL Communications Area, which contains control information for exception handling.

alternate index file. A file that has a different key path over a base file. The base file can be a keyed, direct, or sequential file.

API. Application Programming Interface

array. An object consisting of an ordered collection of homogeneous objects mapped onto N dimensions.

attribute. An object that specifies information about another object, such as the length of a character string, field or the date at which a record was last accessed.

Backus Naur Form. BNF

BNF. The metalanguage, Backus Naur Form.

case. An ordered collection of selections for the declaration of a field.

CCS. Common Communication Support.

CCSID. Coded Character Set Identifier.

CDRA. Character Data Representation Architecture.

character string. A string of bytes containing characters encoded as specified by its CCSID attribute.

CM. Communications Manager

complete path name. The specifications for a file which includes the drive (if OS/2), directory, filename and file extension.

constructor. A data type that consists of zero or more instances of other data types. ADL examples of constructors are ARRAYs, CASEs, AND SEQUENCESs.

CPGID. Code Page Global Identifier.

CTOK. Condition Token. A 12-byte area in which information about the execution of a called program is returned by that program.

CUA. Common User Access.

data conversion. A set of programs that convert data according to defined data descriptions. For example, characters can be converted from EBCDIC to ASCII, and numeric data can be converted from System /370 packed decimal to IEEE floating point or ASCII character (or vice versa).

data description. Specification of the layout of data. The data description of data stored in a file can be viewed as a file attribute.

data security. The protection of data against unauthorized disclosure, transfer, modifications or destruction, whether accidental or intentional.

data set. The major unit of data storage and retrieval. It consists of a collection of data in one of several prescribed arrangements which is described by control information that the system has access to.

data stream. All data transmitted through a data channel in a single read or write operation.

DBCS. Double-byte character set. A set of characters in which each character is represented by 2 bytes.

DD&C. Data Description and Conversion. An architecture extension to DDM.

DDM. A set of interfaces that gives users access to data files that reside on remote systems connected by a communication network. The DDM interfaces enable an application program to retrieve, add, update and delete data records in a file existing on a remote system. The DDM interfaces can be used to communicate between systems that have different architectures.

deadlock. Unresolved contention for the use of a resource. Each element in a process is waiting for an action by, or a response from, the other.

declaration. An ADL statement specifying the type, attributes, and entities of a record or an object.

DFM client. Translates requests from the source system for access to file data on a remote system into a standard architected DDM request.

DFM server. A DFM component that accepts remote requests to access data and translates the requests into data management requests on the target system.

direct file. A file that is organized so that there is a relationship between the contents of the records and their positions.

discriminant. A field that can be tested by a WHEN statement of a CASE to determine if the data declaration clause of the WHEN statement is to be selected.

Distributed Data Management (DDM). Architecture for accessing distributed data located in files and distributed relational databases.

Distributed File Management (DFM). Strategy for a set of programming facilities that implement the file aspects of the DDM architecture on those systems which represent distributed environments.

intersystem communication. Communication between different systems by means of SNA facilities.

DRBA. Distributed relational data base access.

element. An instance of a data type that is a component of a constructor data type.

entity. A record or an object.

fixed-point number. An object representing a number whose precision and scale are fixed.

floating-point number. An object representing a number with fixed precision and floating scale.

FSD. File System Driver.

HLL. High Level Language

HPFS. High Performance File System.

IFS. Installable File System.

keyed file. A file organization that supports keyed access to the records of the file.

LAN. Local Area Network.

Local Area Network. LAN

LDM. Local Data Management.

LDMI. Local Data Management Interface.

local file. A file that resides on the same system as the application program that is accessing it.

LU. Logical unit.

mixed-character string. A character string consisting of both SBCS and DBCS characters.

module. A set of data declarations and plans used to convert data.

object. An instance of a type, such as a field of a record or an attribute of a field.

PL. Programming Language

plan. A program for converting data from one representation to another.

protocol. A set of rules to be followed by communication systems.

RACF. Resource Access Control Facility. An external security management facility.

record. The basic unit of data stored in a file and transferred between DDM source and target servers. An instance of a field or constructor type.

record file. Record files consist of data fields organized into records that can be accessed as a set of bytes.

remote file. A file that resides on a system other than the system where the application program requesting access to the file resides.

Remote Record Access Support. The DFM function that allows VSAM applications to access remote file data.

SBCS. Single-byte character set. A set of characters in which each character is represented by 1 byte.

SCM. Source Communications Manager. The DDM layer responsible for interfacing with the local communications facilities. It coordinates the sending and receiving of data on the source system.

sequence. An object consisting of an ordered collection of heterogeneous objects.

sequential file. A file in which records are arranged in exactly the same sequence as they were stored into the file.

SNA. Systems Network Architecture.

source system. A system that requests access to data on another system. In a client/server relationship, it is the client system.

Stream Agent. The DDM program responsible for transformation of data between the stream oriented API requests and the DDM byte requests.

subtype. In the type hierarchy, a lower level type which inherits characteristics and attributes from a higher level type.

supertype. In the type hierarchy, a higher level type from which a subtype inherits its characteristics and attributes.

Systems Network Architecture (SNA). The description of the logical structure, formats, protocols, and operational sequences for transmitting information units through and controlling the configuration and operation of networks.

target system. The system that contains data that is being accessed by another system. In a client/server relationship, it is the server system.

target system data. Data considered to be owned and maintained according to the rules and functions prescribed by the data manager on the target system.

TP. Transaction Program

user exit. A point in an IBM-supplied program at which a user-exit routine may be given control.

type. A set of representable values encapsulated by a set of operations on those values. Each programming language defines its own set of data types, such as the *PIC* data types of COBOL or the *integer* data type of C.

type domain. The set of programs and objects that process or store data of the same set of programming language data types and representations. Examples of type domains are MVS COBOL and OS/2 C.

type manager. A facility for a single type domain capable of:

- Mapping programming language data descriptions into ADL.
- Mapping the ADL of another type domain into ADL of its own type domain.
- Mapping ADL into programming language data descriptions.

VTAM. Virtual Telecommunications Access Method.

Index

Communicating Your Comments to IBM

SMARTdata UTILITIES
Data Description and Conversion

Publication No. SC26-7091-01

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

If you are mailing a readers' comment form (RCF) from a country other than the United States, you can give the RCF to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF at the back of this book.
- If you prefer to send comments by FAX, use this number:
  - United States: 1-800-426-6209
  - Other countries: (+1)+408+256-7896
- If you prefer to send comments electronically, use this network ID:
  - IBMLink from U.S. and IBM Network: STARPUBS at SJEVM5
  - IBMLink from Canada: STARPUBS at TORIBM
  - IBM Mail Exchange: USIB3VVD at IBMMAIL
  - Internet: starpubs@vnet.ibm.com

Make sure to include the following in your note:

- Title and publication number of this book
- Page number or topic to which your comment applies.

Readers' Comments — We'd Like to Hear from You

SMARTdata UTILITIES
Data Description and Conversion

Publication No. SC26-7091-01

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	□	□	□	□	□

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	□	□	□	□	□
Complete	□	□	□	□	□
Easy to find	□	□	□	□	□
Easy to understand	□	□	□	□	□
Well organized	□	□	□	□	□
Applicable to your tasks	□	□	□	□	□

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? □ Yes □ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name

Address

Company or Organization

Phone No.

Readers' Comments — We'd Like to Hear from You
SC26-7091-01

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
RCF Processing Department
G26/050
5600 Cottle Road
SAN JOSE, CA 95193-0001

Fold and Tape **Please do not staple** Fold and Tape

Cut or Fold
Along Line

IBM

Program Number: 5765-548
5765-549
5622-793
5622-794
5639-B92

Printed in U.S.A.

SC26-7091-01