

CA Culprit™ for CA IDMS™

User Modules Guide

Release 18.5.00, 2nd Edition



This Documentation, which includes embedded help systems and electronically distributed materials, (hereinafter referred to as the "Documentation") is for your informational purposes only and is subject to change or withdrawal by CA at any time. This Documentation is proprietary information of CA and may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of CA.

If you are a licensed user of the software product(s) addressed in the Documentation, you may print or otherwise make available a reasonable number of copies of the Documentation for internal use by you and your employees in connection with that software, provided that all CA copyright notices and legends are affixed to each reproduced copy.

The right to print or otherwise make available copies of the Documentation is limited to the period during which the applicable license for such software remains in full force and effect. Should the license terminate for any reason, it is your responsibility to certify in writing to CA that all copies and partial copies of the Documentation have been returned to CA or destroyed.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CA PROVIDES THIS DOCUMENTATION "AS IS" WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT WILL CA BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST INVESTMENT, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The use of any software product referenced in the Documentation is governed by the applicable license agreement and such license agreement is not modified in any way by the terms of this notice.

The manufacturer of this Documentation is CA.

Provided with "Restricted Rights." Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in FAR Sections 12.212, 52.227-14, and 52.227-19(c)(1) - (2) and DFARS Section 252.227-7014(b)(3), as applicable, or their successors.

Copyright © 2014 CA. All rights reserved. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

CA Technologies Product References

This document references the following CA Product:

- CA Culprit™ for CA IDMS™

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At <http://ca.com/support>, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Providing Feedback About Product Documentation

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at <http://ca.com/docs>.

Documentation Changes

The following documentation updates were made for the 18.5.00, 2nd Edition release of this documentation:

- [What You Can Do with CULLUS45](#) (see page 96)—Added a note about the hex value of 'ććć'.

Contents

Chapter 1: Introduction 11

Overview	11
Types of CA Culprit User Modules.....	12
Summary of CA-supplied User Modules	14
Syntax Diagram Conventions	15

Chapter 2: Input Modules 19

What Is an Input Module?	19
What You Can Do with an Input Module	19
How to Invoke an Input Module.....	20
Processing Spanned Records —z/VSE (CULSPAN)	20
What You Can Do	20
How to Use CULSPAN	20
Helpful Hints.....	25
Selective Retrieval of VSAM files (CULLVSAM)	26
What You Can Do	26
How CULLVSAM Works	27
How to Use CULLVSAM	27
Performing a Sequential Read from a Pointed Start.....	28
Coding the KEY Control Statement for a Pointed Start	28
Coding the ADR Control Statement for a Pointed Start	29
Example—KSDS Pointed Start.....	31
Performing a Direct Read.....	32

Chapter 3: Procedure Modules 37

What Is a Procedure Module?	37
What You Can Do with a Procedure Module	38
What You Can Do with CA-supplied Procedure Modules	38
What a Procedure Module Does	39
How to Invoke a Procedure Module.....	39
Calling a Procedure Module	40
Branching to a Procedure Module	41
Helpful Hints.....	41
The Universal Interface (CULLUS00)	42
What You Can Do	42
How CULLUS00 Works	42

How to Use CULLUS00	43
Dynamic Sequential File Processing (CULLUS01)	44
What You Can Do	44
How to Use CULLUS01	44
Helpful hints.....	46
System Time and Date Retrieval (CULLUS10)	47
What You Can Do	47
How to Use CULLUS10	48
Helpful Hints.....	49
Julian Date Conversion (CULLUS11)	53
What You Can Do	53
How to Use CULLUS11	54
Helpful Hints.....	54
Century Date Conversion (CULLUS12)	55
What You Can Do	56
How to Use CULLUS12	56
Helpful Hints.....	57
Gregorian Date Conversion (CULLUS14)	58
What You Can Do	58
How to Use CULLUS14	59
Helpful Hint.....	60
Universal Date Conversion (CULLUS15)	62
What You Can Do	62
How to Use CULLUS15	62
Helpful Hints.....	64
Random Access of ISAM Files (CULLUS22)	65
What You Can Do	65
How CULLUS22 Works	66
How to Use CULLUS22	66
Helpful Hints.....	68
Source Code Modifications.....	69
Random Access of VSAM Files (CULLUS25)	74
What You Can Do	74
How to Use CULLUS25	75
Helpful Hints.....	77
Creating a Vertical Hexadecimal Dump (CULLUS29).....	78
What You Can Do	78
How to Use CULLUS29	79
Helpful Hints.....	80
Obtaining Hexadecimal Representation (CULLUS31)	81
What You Can Do	81
How to Use CULLUS31	81

Converting Packed Decimal to Binary (CULLUS33)	82
What You Can Do	82
How to Use CULLUS33	83
Helpful Hints.....	84
Converting Packed Decimal to Zoned Decimal (CULLUS34)	84
What You Can Do	84
How to Use CULLUS34	85
Interpreting Bit Settings (CULLUS35).....	86
What You Can Do	86
How to Use CULLUS35	86
Helpful Hints.....	87
Converting Floating Point Values to Packed Decimal (CULLUS36)	88
What You Can Do	88
How to Use CULLUS36	89
Helpful Hints.....	90
Converting Doubleword Binary to Packed Decimal (CULLUS37)	91
What You Can Do	91
How to Use CULLUS37	92
Sending Messages (CULLUS40).....	92
What You Can Do	92
How to Use CULLUS40	93
Moving Fields to an Input Buffer Area (CULLUS43)	94
What You Can Do	94
How to Use CULLUS43	94
Helpful Hint.....	95
Moving Variable-length Data (CULLUS45)	96
What You Can Do	96
How to Use CULLUS45	96
Helpful Hints.....	97
String Search (CULLUS46)	100
What You Can Do	100
How to Use CULLUS46	101
Helpful Hints.....	102
Creating a Run-time Message (CULLUS48)	104
What You Can Do	104
How to Use CULLUS48	104
Converting Binary Strings (CULLUS50)	105
What You Can Do	105
How to Use CULLUS50	105
Concatenating Fields (CULLUS53)	107
What You Can Do	107
How to Use CULLUS53	107

Helpful Hints.....	108
Searching a Table (CULLUS62)	110
What You Can Do	110
How to Use CULLUS62	110
Helpful Hints.....	111
Processing Data Dictionary Reporter Tables (CULLUS64)	113
What You Can Do	113
How to Use CULLUS64	114
Helpful Hints.....	115
Memory Dump (CULLUS99)	116
How to Use CULLUS99	117

Chapter 4: Output Modules 119

What Is an Output Module?.....	119
What You Can Do with an Output Module	119
How to Invoke an Output Module	120
Formatting a Hexadecimal Buffer Dump (CULEDUMP)	120
What You Can Do	121
How to Use CULEDUMP	121
Helpful Hints.....	122
Printing Labels (CULELABL)	123
What You Can Do	123
How to Use CULELABL	124
Printing Multiple Lines (CULEMLIN)	126
What You Can Do	126
How It Works	127
How to Use CULEMLIN	128
Helpful Hints.....	130
Writing Formatted Records to a VSAM File (CULEVSAM)	136
What You Can Do	136
How to Use CULEVSAM	136
Helpful Hints.....	137
Segmenting Reports in a VSE/POWER Run (CULEPOWER)	138
What You Can Do	138
How to Use CULEPOWER as a CA Culprit Output Module	138
Helpful Hints.....	139
How to Use CULEPOWER as a Subroutine	139
Helpful Hints.....	141

Chapter 5: Writing User Modules 145

What You Can Do	145
-----------------------	-----

General Considerations for User-written Modules.....	146
How to Link-edit User Modules	146
Establishing Linkage to a COBOL Module	147
Establishing Linkage to an Assembler Module.....	148
Establishing Linkage to a PL/I Module.....	149
Establishing linkage to a FORTRAN module.....	150
How to Write Input Modules	150
What You Can Do	150
How Information Is Passed.....	150
Coding a COBOL Input Module.....	154
Coding an Assembler Input Module	158
Coding a PL/I Input Module.....	160
How to Write Procedure Modules	161
What You Can Do	161
How Information Is Passed.....	161
Coding a COBOL Procedure Module.....	162
Coding an Assembler procedure module.....	163
Coding a PL/I Procedure Module.....	164
Coding a FORTRAN procedure module	164
Helpful hints.....	164
How to Write Output Modules	165
What You Can Do	165
How Information Is Passed.....	165
Coding a COBOL Output Module.....	168
Coding an Assembler Output Module	170
Coding a PL/I Output Module	171

Chapter 1: Introduction

This section contains the following topics:

[Overview](#) (see page 11)

[Types of CA Culprit User Modules](#) (see page 12)

[Summary of CA-supplied User Modules](#) (see page 14)

[Syntax Diagram Conventions](#) (see page 15)

Overview

This manual provides information about:

- CA-supplied user modules that can be invoked to perform tasks that fall beyond the scope of standard CA Culprit report processing
- Customized user modules that can be written to perform site-specific processing for CA Culprit reports

A basic knowledge of CA Culprit coding techniques is assumed throughout this manual:

- Users familiar with CA Culprit coding techniques can use this manual as a reference for invoking CA-supplied user modules.
- Experienced CA Culprit users with programming backgrounds in COBOL, FORTRAN, PL/I, or Assembler can use this manual as a guide for writing customized modules not available on the CA Culprit installation media.

Types of CA Culprit User Modules

CA Culprit user modules are subroutines that perform tasks beyond the scope of standard CA Culprit processing. The CA Culprit installation media includes a wide range of user modules, which are summarized at the end of this section. Site-specific routines can also be written, stored, and called from CA Culprit code. Instructions for developing your own user modules can be found in "Writing User Modules" chapter.

Before invoking or writing user modules, you should be familiar with CA Culprit coding techniques. Refer to the *CA Culprit for CA IDMS User Guide* and to the *CA Culprit for CA IDMS Reference Guide* for detailed information on coding CA Culprit parameters.

A user module can be one of three types:

- The **input module**, which reads an input file by using information supplied on an INPUT parameter.

An input module is processed during the extract phase (CULL step, as shown in the diagram below) of a CA Culprit job to read one or more files, manipulate input data, and build the CA Culprit input buffer.

- The **procedure module**, which performs type 7 processing on user-supplied data and returns the processing results to user-defined fields.

A procedure module is processed during the extract phase (CULL step) of a CA Culprit job.

- The **output module**, which creates an output file or report formatted to user specifications.

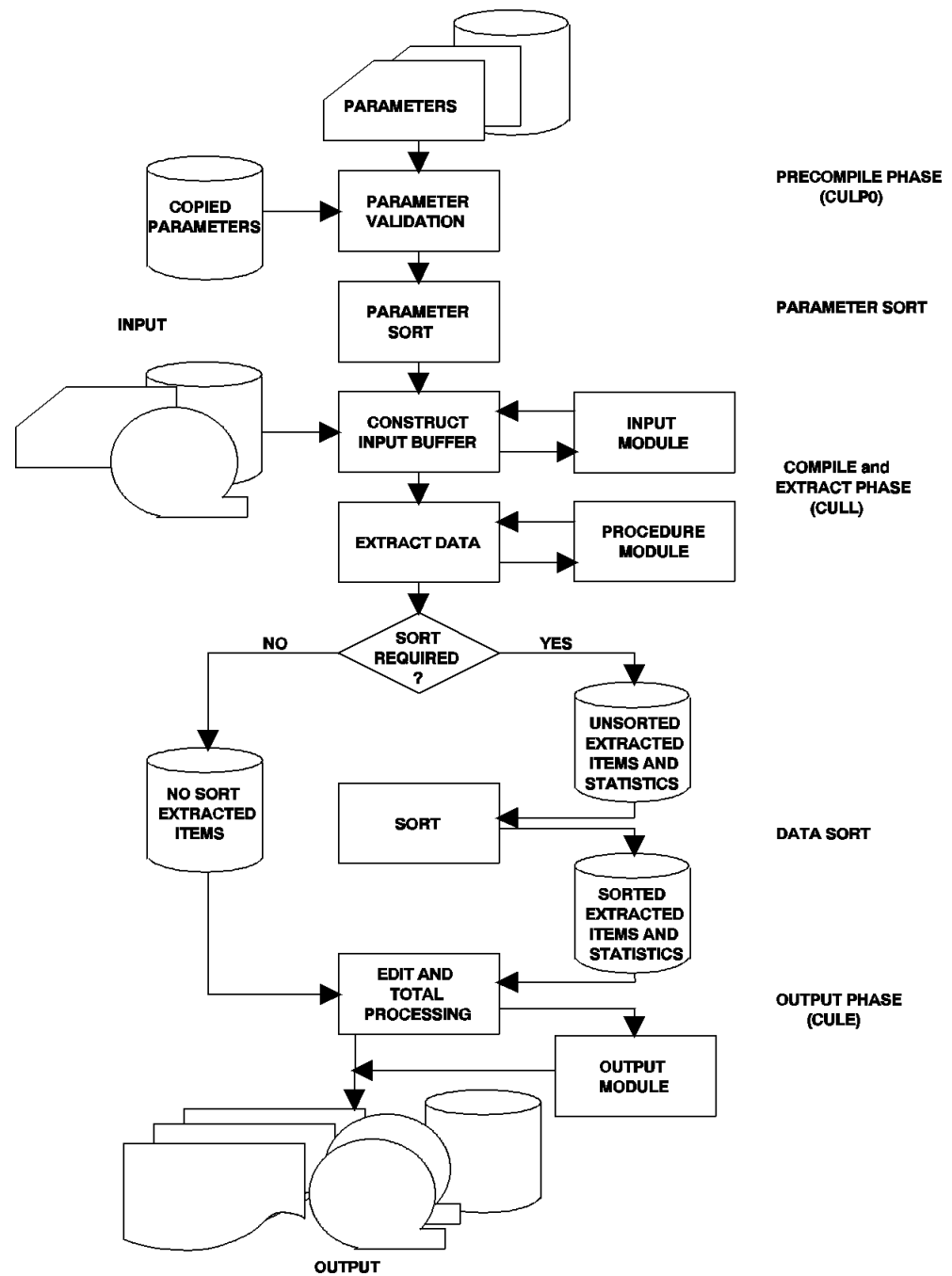
An output module is processed during the output phase (CULE step) of a CA Culprit job, as shown below.

CA-supplied and user-written input, procedure, and output modules are discussed in detail in the following sections.

CA Culprit System Diagram

Input and procedure modules are processed only during the CULL step.

Output modules are processed only during the CULE step.



Summary of CA-supplied User Modules

Input Modules

The module name	What it does
CULSPAN	Reads a spanned record input file (z/VSE)
CULLVSAM	Performs direct reads of key- or entry-sequenced VSAM files

Procedure Modules

The module name	What it does
CULLUS00	Interfaces CA Culprit with user-written modules
CULLUS01	Processes sequential files (z/OS and z/VM)
CULLUS10	Retrieves the system time and date
CULLUS11	Converts a Julian date to Gregorian
CULLUS12	Converts any century date to a user-specified format.
CULLUS14	Converts a Gregorian date to Julian
CULLUS15	Converts a date in any format to a user-specified format
CULLUS22	Retrieves ISAM files
CULLUS25	Retrieves a VSAM file
CULLUS29	Formats a vertical hexadecimal dump
CULLUS31	Displays fields in hexadecimal representation
CULLUS33	Converts packed decimal to binary
CULLUS34	Converts packed decimal to zoned decimal
CULLUS35	Represents bit settings in display format
CULLUS36	Converts floating point values to decimal integers
CULLUS37	Converts doubleword binary to packed decimal
CULLUS40	Sends messages to the console operator (z/VSE)
CULLUS43	Moves variable-length data
CULLUS45	Performs multiple move operations on data
CULLUS46	Performs a character search

The module name	What it does
CULLUS48	Writes a user-written message to the Run-Time Messages Section of a CA Culprit job
CULLUS50	Converts a binary string to a string of characters or work fields
CULLUS53	Concatenates fields
CULLUS62	Searches a CA Culprit table for specified fields
CULLUS64	Maintains a table of user-defined attributes for Data Dictionary Reporter (DDR) reports external to CA Culprit
CULLUS99	Causes a memory dump

Output Modules

The module name	What it does
CULEDUMP	Prints an output line in vertical or horizontal dump format
CULELABL	Creates labels
CULEMLIN	Prints multiple output lines and multiple logical footer lines
CULEVSAM	Writes records to a user-defined VSAM file
CULEPOWR	Segments reports in a CA Culprit job through VSE/POWER

Syntax Diagram Conventions

The syntax diagrams presented in this guide use the following notation conventions:

UPPERCASE OR SPECIAL CHARACTERS

Represents a required keyword, partial keyword, character, or symbol that must be entered completely as shown.

lowercase

Represents an optional keyword or partial keyword that, if used, must be entered completely as shown.

italicized lowercase

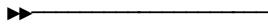
Represents a value that you supply.

lowercase bold

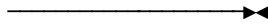
Represents a portion of the syntax shown in greater detail at the end of the syntax or elsewhere in the document.



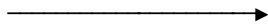
Points to the default in a list of choices.



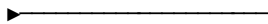
Indicates the beginning of a complete piece of syntax.



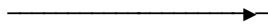
Indicates the end of a complete piece of syntax.



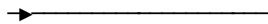
Indicates that the syntax continues on the next line.



Indicates that the syntax continues on this line.



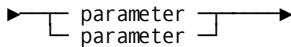
Indicates that the parameter continues on the next line.



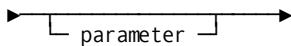
Indicates that a parameter continues on this line.



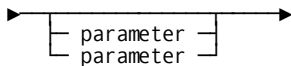
Indicates a required parameter.



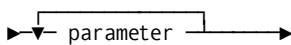
Indicates a choice of required parameters. You must select one.



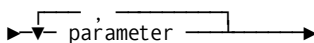
Indicates an optional parameter.



Indicates a choice of optional parameters. Select one or none.



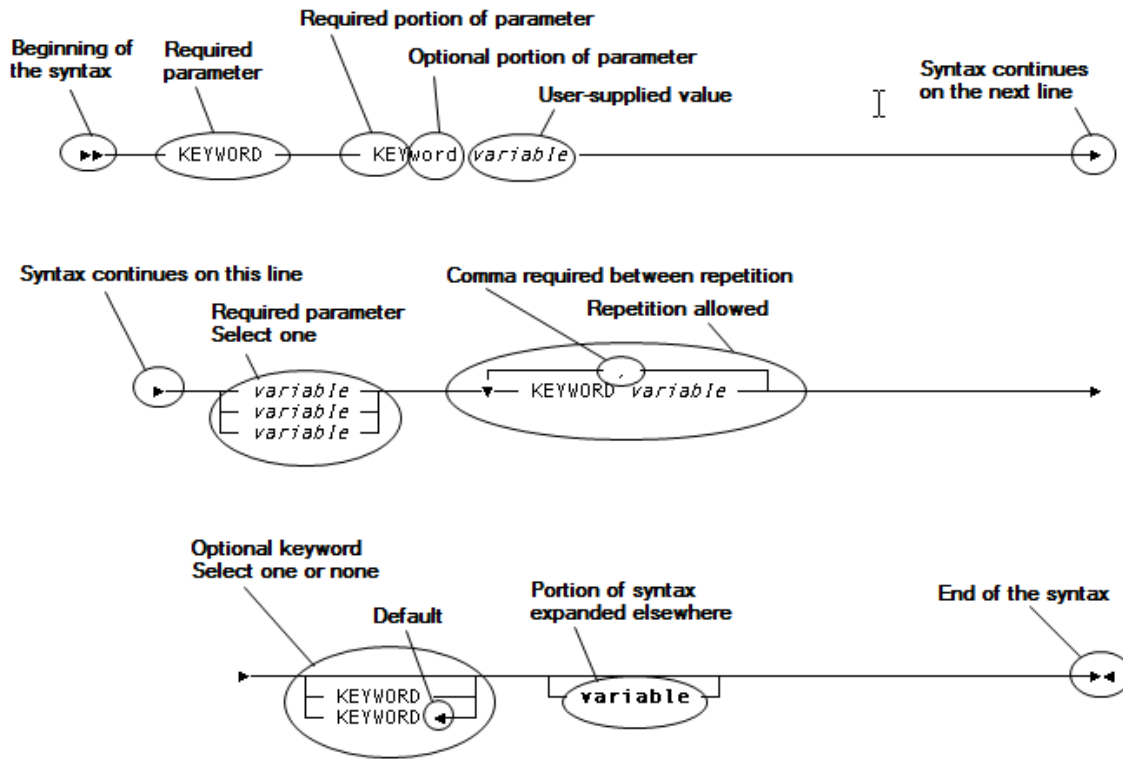
Indicates that you can repeat the parameter or specify more than one parameter.



Indicates that you must enter a comma between repetitions of the parameter.

Sample Syntax Diagram

The following sample explains how the notation conventions are used:



Chapter 2: Input Modules

This section contains the following topics:

[What Is an Input Module?](#) (see page 19)

[What You Can Do with an Input Module](#) (see page 19)

[How to Invoke an Input Module](#) (see page 20)

[Processing Spanned Records —z/VSE \(CULSPAN\)](#) (see page 20)

[Selective Retrieval of VSAM files \(CULLVSAM\)](#) (see page 26)

What Is an Input Module?

An input module is a subroutine called from a CA Culprit INPUT parameter to read files not normally available to a standard CA Culprit run.

Two input modules, CULSPAN and CULLVSAM, are supplied on CA Culprit installation media and described in this section.

What You Can Do with an Input Module

The tasks you can perform with CA-supplied input modules are listed as follows.

To...	Use...
Read an input file containing spanned records (z/VSE only)	CULSPAN
Read key-sequenced (KSDS) or entry-sequenced (ESDS) VSAM files sequentially from a pointed start	CULLVSAM
Read KSDS or ESDS VSAM files directly by key or relative byte address (RBA)	

How to Invoke an Input Module

You can invoke an input module by naming the module on the UM option of the INPUT parameter:

Col
2
↓
INPUT [*record-size-n* [*E*
V
U] [*block-size-n*] **UM** (*module-name*)

Processing Spanned Records—z/VSE (CULSPAN)

What You Can Do

If you are working in a z/VSE environment, you can use CULSPAN to read an input file containing records that span one or more blocks. CULSPAN delivers the records to the CA Culprit input buffer and allows you to use standard CA Culprit code for further processing.

How to Use CULSPAN

To invoke CULSPAN:

1. **Define the z/VSE input file** in the FILE SECTION of the CULSPAN source code.

The source code for CULSPAN is written in COBOL, as shown in the following figure. Instructions for defining the input file in the source code follow the COBOL listing.

2. **Compile and link edit** the CULSPAN code.
3. **Code an INPUT parameter** with:

- The size of the largest input record
- V, for variable-length records
- The block size of the input file
- The module name

Col
2
↓
INPUT *maximum-record-size-n* *V* *block-size-n* **UM** (*CULSPAN*)

CULSPAN Source Code Listing

CULSPAN is supplied in source form only. Each site must adapt CULSPAN for its use by changing the COBOL statements that describe the input file.

```

001000 IDENTIFICATION DIVISION.

002500*   CONTAINS PTF # 83-01-0076   TJG 15:20:19 07/28/83
003000 PROGRAM-ID.      CULSPAN.

005000 AUTHOR.          CA, Inc.

009000 REMARKS.          THIS PROGRAM IS AN INPUT MODULE WHICH
010000                   READS AN INPUT REC INTO CULPRIT'S INPUT BUFFER.

012000 DATE-WRITTEN.    mm/dd/yy
013000 DATE-COMPILED.  mm/dd/yy

045000 ENVIRONMENT DIVISION.

047000 CONFIGURATION SECTION.
048000 SOURCE-COMPUTER.  IBM-370.
049000 OBJECT-COMPUTER.  IBM-370.

051000 INPUT-OUTPUT SECTION.
052000 FILE-CONTROL.

070000 DATA DIVISION.

072000 FILE SECTION.
073000 FD   SPANNED-FILE
074000       BLOCK CONTAINS 2000 CHARACTERS
075000       LABEL RECORD IS LABEL-RECORD
076000       RECORD CONTAINS 187 TO 339 CHARACTERS
077000       RECORDING MODE IS S
078000       DATA RECORD IS SPANNED-RECORD.
079000 01  LABEL-RECORD          PICTURE X(80).
080000 01  SPANNED-RECORD        PICTURE X(339).

082000 WORKING-STORAGE SECTION.

084000 01  SELECT-SWITCH          PICTURE X(1) VALUE ' '.
085000 01  SWITCH-VALUES.
086000       02  FILE-CLOSE-STATUS PICTURE X(1) VALUE ' '.
087000       02  FILE-OPEN-STATUS  PICTURE X(1) VALUE ' '.
088000       02  FILE-STOP-STATUS  PICTURE X(1) VALUE ' '.
089000* *****
090000* *
091000* NOTE: INFORMATIONAL *
092000* *
093000* THE ABOVE THREE SWITCH VALUES ARE MULTIPUNCHED CODES. *
094000* THEY ARE AS FOLLOWS: *
095000*           CLOSE IS HEX'FF' MULTI=12,11,0,7,8,9 *
096000*           OPEN  IS HEX'00' MULTI=12,0,1,8,9 *
097000*           STOP  IS HEX'0F' MULTI=12,7,8,9 *
098000* *****
099000 01  ERROR-MESSAGES.
100000       02  ERROR-MSG1          PICTURE X(37)
101000           VALUE 'CULSPAN ERROR - INVALID CULARG SWITCH'.
102000       02  ERROR-MSG2          PICTURE X(31)
103000           VALUE 'CULSPAN ERROR - CULARG SWITCH= '.

105000 LINKAGE SECTION.
106000 01  CULARG-INPUT          PICTURE X(80).
107000 01  CULARG-DEVICE-CODE    PICTURE X(1).
108000 01  CULARG-SWITCH          PICTURE X(1).
109000 01  CULARG-FORMAT-CODES    PICTURE X(2).
110000 01  CULARG-RECORD-SIZE    PICTURE 9(2) USAGE COMP.
111000 01  CULARG-BLOCK-SIZE     PICTURE 9(2) USAGE COMP.
112000 01  CULARG-FILE-NAME       PICTURE X(8).
113000 01  CULARG-DO-NOT-USE      PICTURE X(1).
114000 01  CULARG-PRINT-ROUTINE   PICTURE X(1).
116000 PROCEDURE DIVISION
118000     USING CULARG-INPUT
119000           CULARG-DEVICE-CODE
120000           CULARG-SWITCH
121000           CULARG-FORMAT-CODES
122000           CULARG-RECORD-SIZE

```

```

123000      CULARG-BLOCK-SIZE
124000      CULARG-FILE-NAME
125000      CULARG-DO-NOT-USE
126000      CULARG-PRINT-ROUTINE.
127000
129000 PARA01-CULSPAN-CONTROL.
130000      MOVE ' ' TO SELECT-SWITCH.
131000      IF CULARG-SWITCH = FILE-CLOSE-STATUS
132000          PERFORM PARA02-OPEN THRU PARA02-EXIT
133000      ELSE
134000          IF CULARG-SWITCH = FILE-OPEN-STATUS
135000              PERFORM PARA03-READ THRU PARA03-EXIT
136000              UNTIL SELECT-SWITCH = 'Y'
137000          ELSE
138000              IF CULARG-SWITCH = FILE-STOP-STATUS
139000                  PERFORM PARA05-CLOSE THRU PARA05-EXIT
140000              ELSE
141000                  PERFORM PARA06-SWITCH-ERROR
142000                  THRU PARA06-EXIT.
143000
144000      GOBACK.

146000 PARA02-OPEN.
147000      OPEN INPUT SPANNED-FILE.
148000      MOVE FILE-OPEN-STATUS TO CULARG-SWITCH.
149000      PERFORM PARA03-READ THRU PARA03-EXIT
150000      UNTIL SELECT-SWITCH = 'Y'.
151000 PARA02-EXIT.
152000      EXIT.

154000 PARA03-READ.
155000      READ SPANNED-FILE INTO CULARG-INPUT
156000      AT END
157000          PERFORM PARA04-CLOSE THRU PARA05-EXIT
158000          MOVE 'Y' TO SELECT-SWITCH
159000          GO TO PARA03-EXIT.
160000 PARA03-SELECT.
161000      CALL 'CULLCBSL' USING CULARG-INPUT SELECT-SWITCH.
162000 PARA03-EXIT.
163000      EXIT.

165000 PARA04-CLOSE.
166000      MOVE FILE-CLOSE-STATUS TO CULARG-SWITCH.

168000 PARA05-CLOSE.
169000      CLOSE SPANNED-FILE.
170000 PARA05-EXIT.
171000      EXIT.

173000 PARA06-SWITCH-ERROR.
174000      DISPLAY ERROR-MSG1.
175000      DISPLAY ERROR-MSG2 CULARG-SWITCH.
176000      PERFORM PARA04-CLOSE THRU PARA05-EXIT.
177000 PARA06-EXIT.
178000      EXIT.

```

CULSPAN Source Code Modifications

The z/VSE input file must be defined in the FILE SECTION of the CULSPAN source code, as shown below:

```
DATA DIVISION.  
  
FILE SECTION.  
FD  SPANNED-FILE  
   BLOCK CONTAINS number-of-characters CHARACTERS  
   LABEL RECORD IS label-record-name  
   RECORD CONTAINS minimum-record-size-n TO  
   maximum-record-size-n CHARACTERS  
   RECORDING MODE IS S  
   DATA RECORD IS SPANNED-RECORD.  
01  LABEL-RECORD          PIC X(80).  
01  SPANNED-RECORD        PIC X(maximum-record-size-n).
```

- *Number-of-characters* specifies the maximum number of characters contained in an input file block.
- *Label-record-name* identifies the label record on the input file:
 - Nonstandard or user-supplied label records are coded on a 01 level.
 - Standard or omitted label records are coded with an appropriate COBOL clause, such as LABEL RECORDS ARE STANDARD or LABEL RECORDS ARE OMITTED.
- *Minimum-record-size-n* TO *maximum-record-size-n* specifies minimum and maximum record sizes on the input file. The maximum record size on the SPANNED-RECORD description must match the record size entered in the RECORD CONTAINS clause.

Helpful Hints

- z/OS users can read spanned records by specifying RECFM=VBS and BFTEK=A subparameters in the JCL statement that defines the input file.

```
//SYS010 DD DSN=user.inputfil,UNIT=tape,DISP=SHR,  
           VOL=SER=nnnnnn,  
           DCB=(RECFM=VBS,BFTEK=A,LRECL=256,BLKSIZE=3156)
```

user.inputfil = data set name for primary input file
tape = symbolic device name for tape file
nnnnnn = volume serial number

- Spanned records are not supported for z/VM.

Example

This sample code shows the COBOL source code and the CA Culprit INPUT parameter needed to read spanned input records in a z/VSE environment.

In the following code:

- The spanned-file is assigned to SYS010 in the COBOL source code FILE-CONTROL section.
- SYS010 is a CA Culprit default and does not have to be specified in the CA Culprit code.
- Record size ranges from 187 to 339 characters.
- Block size is 2000 characters.

The modified COBOL source code:

```
.  
.   
.   
  
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
    SELECT SPANNED-FILE ASSIGN TO SYS010-UT-3420-S.  
  
DATA DIVISION.  
  
FILE SECTION.  
FD  SPANNED-FILE  
    BLOCK CONTAINS 2000 CHARACTERS  
    LABEL RECORD IS LABEL-RECORD  
    RECORD CONTAINS 187 to 339 CHARACTERS  
    RECORDING MODE IS S  
    DATA RECORD IS SPANNED-RECORD.  
01  LABEL-RECORD                PIC X(80).  
01  SPANNED-RECORD            PIC X(339).  
.  
.  
.
```

The CA Culprit code:

```
Col  
2  
▼  
IN 339 V 2000 UM(CULSPAN)
```

Selective Retrieval of VSAM files (CULLVSAM)

What You Can Do

CULLVSAM retrieves selected records from a VSAM file that is used as input for a CA Culprit run. You can use CULLVSAM to read selected variable- or fixed-length records that are stored in key-sequenced (KSDS) or entry-sequenced (ESDS) VSAM files. The VSAM records can be read:

- **Sequentially** from a particular point in the VSAM file (pointed start)
- **Directly** by key or relative byte address (RBA)

How CULLVSAM Works

- For **sequential reads using a pointed start**, CULLVSAM uses a key control statement to target the first VSAM record for the read. The key control statement specifies:
 - A relative byte address for entry-sequenced files
 - A key for key-sequenced files

After CA Culprit finds the target record, records are delivered sequentially to the input buffer until the end of the file is reached. If the starting record is not found, the read starts at the record with the next highest key.

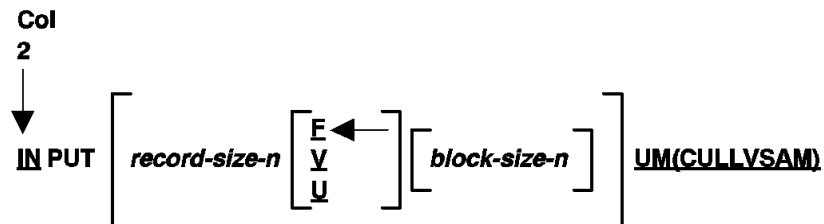
- For **direct reads**, CULLVSAM requires three files:
 - The VSAM source file
 - A sequential file (key file) that contains records with either key values for a KSDS file or relative byte addresses for an ESDS file
 - A VSAM control file to define the key file

Upon execution of CULLVSAM, CA Culprit delivers the key file record to the beginning of the input buffer, followed by the retrieved VSAM record and the RDW, which is a binary field that overlay the last two bytes.

How to Use CULLVSAM

To invoke CULLVSAM, code:

1. An **INPUT** parameter specifying CULLVSAM on the UM option:



- *Record-size-n* is a number in the range 1 through 32767 that specifies the size of the input record.
 - *Block-size-n* is a number in the range 1 through 32767 that specifies the size of a physical block of records.
2. **One or more control statements**, as needed, in the job control language for CA Culprit's extract phase (CULL step). The necessary control statements are described later in this section.

Performing a Sequential Read from a Pointed Start

To read records sequentially from a specific place in the input file, code:

- **An INPUT parameter** using the UM(CULLVSAM) option
- **REC parameters** describing the VSAM file
- **Control statements** defining a password, if necessary, and the starting point for the read:
 - Use **PW=** as the first control statement if the file is password protected.
 - Use the **KEY control statement** for key-sequenced files. See "Coding the KEY control statement for a pointed start" in this section for the syntax.
 - Use the **ADR control statement** for entry-sequenced files. See "ADR control statement" in this section for the syntax.

Coding the KEY Control Statement for a Pointed Start

To access key-sequenced files at a specific place in the file, code the following control statements.

Pointed Start for z/OS

For **z/OS**, assign the external file name VSAMCTRL for the control statements:

```
//VSAMCTRL DD *
KEY key-field-format-a key-value-q
```

Key-field-format-a is a 1-character code in column 6 that specifies the key value format:

The code...	Specifies...
C	Character
H	Hexadecimal

Key-value-q, coded in column 7 and enclosed in single quotation marks, specifies an alphanumeric or hexadecimal value of the target key.

Pointed Start for z/VSE

For **z/VSE**, read the control statement(s) in from SYSIPT:

5-step JCL:

```
// ASSGN SYSIPT,X'device'  
// EXEC CULL,SIZE=300K  
KEY key-field-format-a key-value-q
```

1-step JCL:

```
// EXEC CULPRIT,SIZE=300K  
CULPRIT PARAMETERS  
/*  
KEY key-field-format-a key-value-q  
/*
```

The size specification must be small enough to allow space allocation for VSAM modules in the GETVIS area.

Coding the ADR Control Statement for a Pointed Start

To access entry-sequenced files at a specific place in the file, code the following control statements:

Pointed Start for z/VSE

For **z/VSE**, assign the external file name VSAMCTRL for the control statements:

```
//VSAMCTRL DD *  
ADR H hexadecimal-position-qx
```

- **H** is a 1-character keyword, coded in column 6, that formats the key field.
- *Hexadecimal-position-qx* specifies a full word hexadecimal value (8 digits) that identifies the starting byte of the targeted record within the file. The first byte of the file is always '00000000'. *Hexadecimal-position-qx* starts in column 7 and is enclosed in single quotation marks.

Pointed Start for z/OS

For **z/OS**, read the control statement(s) in from SYSIPT:

5-step JCL:

```
// ASSGN SYSIPT,X'device'  
// EXEC CULL,SIZE=300K  
ADR H hexadecimal-position-qx
```

1-step JCL:

```
// EXEC CULPRIT,SIZE=300K  
CULPRIT PARAMETERS  
/*  
ADR H hexadecimal-position-qx  
/*
```

The size specification must be small enough to allow space allocation for VSAM modules in the GETVIS area.

Notes

- The input buffer size should include two additional bytes for the RDW to avoid any loss of data.
- To read a VSAM file from the beginning, use the VS option of the INPUT parameter. VS allows retrieval from all types of VSAM files.
- Valid control statements must be used in CA Culprit JCL. If CULLVSAM encounters a blank card (z/VSE) or if VSAMCTRL is DUMMY (z/OS) the read will start at the beginning of the file.
- If a file is password protected and the PW= control statement is omitted, the console operator must supply the password for the file to be opened.
- Partial keys, starting with the leftmost character, can be used for accessing key-sequenced files.

Example—KSDS Pointed Start

This example performs a sequential read of a KSDS file, starting with account 7778888.

The following code:

- Specifies CULLVSAM on the UM option of the INPUT parameter
- Instructs CULLVSAM, through the key control statement, to start reading the file with the record that contains 7778888 in the first position of the key field

```
IN 80 F 80 UM(CULLVSAM)
REC NAME          1 20
REC BALANCE       21 6 2 DP=2
REC ACCOUNT       33 4 3
013EXAMPLE OF CULLVSAM
01410038 'SEQUENTIAL READ FROM A POINTED START OF'
01410078 'KEY-SEQUENCED FILE'
01420001 ' '
0151*001 NAME      HH 'NAME'
0151*002 BALANCE   HH 'BALANCE'
0151*003 ACCOUNT   FN HH 'ACCOUNT'
010UT D
//CULPRIT.VSAMCTRL DD *
KEY C'7778888'
```

Example—ESDS Pointed Start

This example does a sequential read of records in an ESDS file, starting with the second record.

The following code:

- Specifies CULLVSAM on the UM option of the INPUT parameter
- Uses the key control statement, coded in the JCL CULL step, to instruct CULLVSAM to start a sequential read from the relative byte address of the second record in the file

```

IN 80 F 80 UM(CULLVSAM)
REC NAME          1 20
REC BALANCE       21 6 2 DP=2
REC ACCOUNT       33 4 3
013EXAMPLE OF CULLVSAM
01410040 'SEQUENTIAL READ FROM A POINTED START OF'
01410080 'ENTRY-SEQUENCED FILE'
01420001 ' '
0151*001 NAME      HH 'NAME'
0151*002 BALANCE   HH 'BALANCE'
0151*003 ACCOUNT   FN HH 'ACCOUNT'
010UT D
// ASSGN SYSIPT,X'device'
// EXEC CULL,SIZE=300K
ADR H'00000050'

```

Performing a Direct Read

To perform a direct read, code the following:

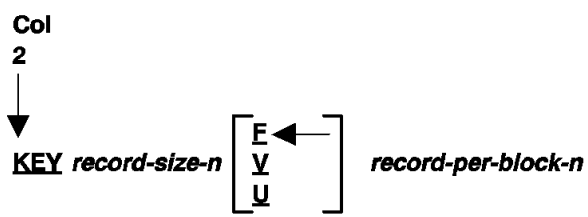
1. The **INPUT parameter** using the UM(CULLVSAM) option
2. **REC parameters** describing the VSAM file

The start position of the VSAM record is relative to the beginning of the input buffer. The input buffer contains the entire key-file record followed by the VSAM record.
3. **Type 7 logic** (for key-sequenced files) to compare key file values to the key value of the retrieved VSAM record
4. The **VSAM and key file assignment** in the CA Culprit job control language:
 - The VSAM file is assigned to SYS010 in the CA Culprit JCL.

If an alternate input file is required, it can be assigned with the DD= option on the INPUT parameter.
 - The sequential file containing key values (a key file) is assigned to SYS002.
5. A **KEY control statement** in the CULL step of the CA Culprit job control language. See the following "Coding the KEY control statement" for syntax.

Coding the KEY Control Statement for Direct Reads

To directly access specific records in the file, code the KEY control statement as shown below:



file type-a label-type-a { $\begin{array}{c} K \\ A \end{array} \}$ start-position-n field-size-n

- **Record-size-n**, coded in columns 5 through 8, is a 4-digit number, that indicates the size of the key file record.
- **F/V/U**, coded in column 9, identifies the record type:

The code...	Identifies...
F (default)	Fixed-length records
V	Variable-length records
U	Undefined length records

- **Records-per-block-n**, coded in columns 10 through 12, is a 3-digit number that indicates the number of records in each block on the key file.
- **File-type-a**, coded in column 13, is a 1-character code that defines the structure of the key file:

The code...	Identifies a...
Blank (default)	Sequential file (z/VSE) Tape file (z/VSE)
4	Punched card (z/VSE)
8	VSAM file (z/OS)

- **Label-type-a**, coded in column 14, is a 1-character code that specifies the label type of the key file:

The code...	Identifies...
Blank (default)	Standard labels
S	Standard labels
N	No labels
A	Standard labels User-defined labels

- **K/A** is a 1-character keyword, coded in column 17, that identifies the VSAM file type:

The keyword...	Identifies...
K	KSDS
A	ESDS

- **Start-position-n**, coded in columns 22 through 25, is a 4-digit number that indicates the starting position of the key value on the key file record.
- **Field-size-n**, coded in columns 26 and 27, is a 2-digit number that indicates the length of the key field. When the key field size on the key file is less than the length of the VSAM key, the key file value is padded with binary zeros on the right.

Notes

- If the file is password protected, it cannot be opened without a valid password. Use PW= as the first control statement or omit the PW= statement and allow the console operator to supply the password.
- If a match between keys on the VSAM file and the key file cannot be made, CULLVSAM will return the record with the next higher key.
- If a record with the next higher key does not exist, CULLVSAM will return two asterisks (**) to the first two positions of the VSAM area in the CA Culprit input buffer.

Example

This example retrieves selected records directly from a variable-length KSDS file.

The following code uses:

- An INPUT parameter that specifies:
 - Variable-length records
 - A record size of 162 bytes, consisting of:
 - The key file record size (80 bytes)
 - The VSAM record size (80 bytes)
 - The record descriptor word (RDW) (2 bytes)
- A key control statement that describes the key file as:
 - A KSDS file having standard labels, 80-character fixed-length records, and a block size of one record
 - Having a key that is 4-bytes long and starts in position 1 of the key file
- Type 7 logic checks the key values of the key file and the VSAM record.

```
IN 162 V 80 UM(CULLVSAM)
REC KEYFILE-KEY  1  4  3
REC NAME          81 16
REC BALANCE       97  6  2  DP=2
REC ACCOUNT      109  4  3
013EXAMPLE OF CULLVSAM
01410040 'DIRECT READ OF A VARIABLE LENGTH KEY-SEQUENCED FILE'
0142*001 ' '
0151*001 NAME          HH 'NAME'
0151*002 BALANCE       HH 'BALANCE'
0151*003 ACCOUNT      FN HH 'ACCOUNT'
0151*004 KEYFILE-KEY  FN HH 'KEY FROM' 'KEYFILE'
017      IF NAME EQ '***' DROP
017      IF KEYFILE-KEY EQ ACCOUNT TAKE
017      DROP
010UT  D
//CULPRIT.VSAMCTRL DD *
KEY0080F001  K 000104
```


Chapter 3: Procedure Modules

This section contains the following topics:

[What Is a Procedure Module?](#) (see page 37)
[What You Can Do with a Procedure Module](#) (see page 38)
[What a Procedure Module Does](#) (see page 39)
[How to Invoke a Procedure Module](#) (see page 39)
[The Universal Interface \(CULLUS00\)](#) (see page 42)
[Dynamic Sequential File Processing \(CULLUS01\)](#) (see page 44)
[System Time and Date Retrieval \(CULLUS10\)](#) (see page 47)
[Julian Date Conversion \(CULLUS11\)](#) (see page 53)
[Century Date Conversion \(CULLUS12\)](#) (see page 55)
[Gregorian Date Conversion \(CULLUS14\)](#) (see page 58)
[Universal Date Conversion \(CULLUS15\)](#) (see page 62)
[Random Access of ISAM Files \(CULLUS22\)](#) (see page 65)
[Random Access of VSAM Files \(CULLUS25\)](#) (see page 74)
[Creating a Vertical Hexadecimal Dump \(CULLUS29\)](#) (see page 78)
[Obtaining Hexadecimal Representation \(CULLUS31\)](#) (see page 81)
[Converting Packed Decimal to Binary \(CULLUS33\)](#) (see page 82)
[Converting Packed Decimal to Zoned Decimal \(CULLUS34\)](#) (see page 84)
[Interpreting Bit Settings \(CULLUS35\)](#) (see page 86)
[Converting Floating Point Values to Packed Decimal \(CULLUS36\)](#) (see page 88)
[Converting Doubleword Binary to Packed Decimal \(CULLUS37\)](#) (see page 91)
[Sending Messages \(CULLUS40\)](#) (see page 92)
[Moving Fields to an Input Buffer Area \(CULLUS43\)](#) (see page 94)
[Moving Variable-length Data \(CULLUS45\)](#) (see page 96)
[String Search \(CULLUS46\)](#) (see page 100)
[Creating a Run-time Message \(CULLUS48\)](#) (see page 104)
[Converting Binary Strings \(CULLUS50\)](#) (see page 105)
[Concatenating Fields \(CULLUS53\)](#) (see page 107)
[Searching a Table \(CULLUS62\)](#) (see page 110)
[Processing Data Dictionary Reporter Tables \(CULLUS64\)](#) (see page 113)
[Memory Dump \(CULLUS99\)](#) (see page 116)

What Is a Procedure Module?

A procedure module is an Assembler, PL/I, COBOL, or FORTRAN subroutine that is called during type 7 processing logic to facilitate special processing tasks performed by CA Culprit.

This section presents a general discussion of procedure modules, followed by a discussion of each CA-supplied module and an example.

What You Can Do with a Procedure Module

The tasks you can perform with CA-supplied procedure modules are listed in the following table.

What You Can Do with CA-supplied Procedure Modules

To...	Use...
Use your own user-written module with CA Culprit	CULLUS00
Retrieve sequential file records during the execution of a CA Culprit run (z/OS)	CULLUS01
Retrieve system time and date	CULLUS10
Convert a Julian date to Gregorian	CULLUS11
Convert century dates to a user-specified format	CULLUS12
Convert a Gregorian date to Julian	CULLUS14
Convert any date format to a user-specified format	CULLUS15
Retrieve ISAM files (direct access)	CULLUS22
Retrieve VSAM files (direct access)	CULLUS25
Format a vertical hexadecimal dump	CULLUS29
Display in hexadecimal	CULLUS31
Convert a packed decimal field to binary	CULLUS33
Convert a packed decimal field to zoned decimal	CULLUS34
Display bit settings	CULLUS35
Convert a floating point value to decimal	CULLUS36
Convert double-word binary to packed decimal	CULLUS37
Send messages to the console operator (z/VSE)	CULLUS40
Move variable-length data from one field to another	CULLUS43
Move variable-length fields to fixed-length locations	CULLUS45
Search a character string	CULLUS46
Generate a run-time message	CULLUS48
Convert a binary string to alphanumeric or numeric values	CULLUS50
Concatenate fields	CULLUS53

To...	Use...
Search a CA Culprit table	CULLUS62
Create and read a table of attributes from the Integrated Data Dictionary	CULLUS64
Cause a memory dump	CULLUS99

What a Procedure Module Does

When a procedure module is called from type 7 logic, CA Culprit:

1. Loads a single copy of the module.
2. Constructs an argument table, which contains:
 - a. The starting address of the CA Culprit input buffer
 - b. Pointers to a maximum of nine user-supplied arguments
3. Passes control to the procedure module. The module then processes the data received through coded module arguments and returns the results to receiving fields defined in the CA Culprit program.
4. Resumes processing control. Processing begins with the statement in type 7 logic immediately following the CALL to the procedure module.

How to Invoke a Procedure Module

You can invoke procedure modules by either:

- Issuing a CALL statement from a type 7 parameter
- Moving data into reserve words (ARG1 through ARG9) and then branching to the module

The CALL statement is the most convenient method and is used throughout this manual. Each method is described as follows.

Calling a Procedure Module

First—**Define input or work fields to hold argument values** that are sent to the procedure module.

Second—**Define input or work fields to receive values** returned from the procedure module.

Usually, type 0 work fields are initialized to spaces (for alphanumeric fields) or zeros (for numeric fields).

Third—**Issue a CALL statement** from a type 7 parameter to the module being invoked:

Col
2



RPT-*nn*7*sss* CALL US*nn* [(*module-argument* . . .)]

- *Rpt-nn*, coded in columns 2 and 3, specifies a 2-digit number in the range 00 through 99 that identifies the CA Culprit report.
- *Sss*, coded in columns 5 through 7, specifies a 3-digit number indicating the sequence number of the type 7 parameter.
- *Nn* specifies a 2-digit number in the range 00 through 99 that identifies the procedure module.
- *Module-argument* specifies one or more values to be passed to and from the module.

All values must be specified in sequence. To omit an alphanumeric argument, specify a blank enclosed in single quotation marks. To omit a numeric value, specify a zero.

Fourth—**Reset argument values** before reissuing a CALL to the same module.

Branching to a Procedure Module

1. Move values individually to the reserved field names (ARG1 through ARG9).
2. Follow the MOVE statements by a branch (B) to the procedure module.

The following example moves values to arguments (ARG) 1 through 4 before branching to US33:

```
Col
2
▼
.
.
.
017010 MOVE PACKED-NUMBER TO ARG1
017020 MOVE 8 TO ARG2
017030 MOVE BINARY-RESULT TO ARG3
017040 MOVE 4 TO ARG4
017050 B US33
```

Helpful Hints

- Arguments must be coded in the sequence shown in the syntax. A zero (numeric) or a space, enclosed in single quotation marks, (alphanumeric) are used as place holders for unused arguments.
- Numeric arguments should be 8-byte packed decimal. Decimal positions and more than 15 digits are not allowed. When using values from work fields:
 - Omit DP= specifications
 - Omit initial values that contain decimals
- You can use up to 100 procedure modules in a single CA Culprit job.
- You can invoke the same procedure module in more than one report in a single CA Culprit run.

The Universal Interface (CULLUS00)

What You Can Do

CULLUS00 acts as an interface between a user-written subroutine that is not written specifically for CA Culprit and a CA Culprit run. You can use CULLUS00 to invoke up to 25 processing subroutines that you have written, providing your routines:

- Omit pointing to the address of the input buffer in the first argument
- Use CULF as a prefix to the name if the module being called is written in FORTRAN.
- Are compiled and link edited

For details on writing and link editing your own modules, see the chapter "Writing User Modules."

How CULLUS00 Works

CULLUS00:

- Dynamically loads the user-written module to make it accessible to CA Culprit.
- Automatically passes up to eight arguments specified immediately after the module name on the CALL statement to the user-written module.

Contrary to CA-supplied modules that do not require the CULLUS00 interface, the address of CA Culprit's input buffer is not passed in the first argument (ARG1).

- Automatically converts CA Culprit numeric work fields to FORTRAN data formats when a FORTRAN subroutine with a CULF name prefix is called:

CULPRIT field...	FORTRAN format...
8-byte packed decimal	4-byte binary field
16-byte packed decimal (8 decimal places assumed)	Double-precision floating point

Conversion to single-precision floating point is not supported by CULLUS00.

How to Use CULLUS00

To invoke CULLUS00:

1. *Define the sending and receiving fields* (arguments) within the CA Culprit program.
2. *Issue a CALL statement* from type 7 logic:

Col
2



RPT-*nn*sss CALL US00 (*module-name-q* [*argument-v...*])

- *Rpt-nn*, coded in columns 2 and 3, specifies a 2-digit number in the range 00 through 99 that identifies the CA Culprit report.
- *Sss*, coded in columns 5 through 7, specifies a 3-digit number indicating the sequence number of the type 7 parameter.
- **Module-name-q** (ARG1) requires an 8-character user-written module name, enclosed in single quotation marks. If the name is less than 8 characters, pad the right with blanks.
- **Argument-v** (ARG2 through ARG9) requires 1 to 8 values, separated by a space or comma, that correspond to the arguments expected by the called module.

Example

This sample code shows the CA Culprit parameters required to call the user-written module MYPROG, which converts Fahrenheit temperatures to centigrade.

The following code:

- Defines work fields for the sending field (FAHREN) and the receiving field (CENT)
- Issues a CALL to CULLUS00 from a type 7 parameter to convert 820 Fahrenheit to centigrade.

```
IN 80 F 2960
REC FIELD1 1 80
010 FAHREN 82
010 CENT 0
013 TEMPERATURE CONVERSION USING CULLUS00
.
.
.
017010 CALL US00 ('MYPROG ' FAHREN CENT)
```

Dynamic Sequential File Processing (CULLUS01)

What You Can Do

CULLUS01 allows z/OS users to dynamically access a sequential data set during a CA Culprit run. Using CULLUS01, you can:

- Open a sequential input file during processing
- Read records from the sequential file into fields defined in the CA Culprit code
- Close the sequential input file before end-of-file has been reached by CULLUS01

How to Use CULLUS01

To invoke CULLUS01:

First—**Define a dummy buffer area** equal to the length of the retrieved record.

- For non-database runs, use the MB= option on an additional INPUT parameter or specify an alphanumeric work field that has the length of the retrieved record.
- For database runs, use one INPUT parameter that includes the extra storage requirement in the record size specification.

Second—**Define fields in the dummy buffer area** by using REC parameters.

Third—**Define a 1-character alphanumeric work field** to contain user instructions and CULLUS01 return values. Valid contents of this field are listed as follows under *task-v*.

Fourth—**Issue a CALL to CULLUS01** in type 7 logic:

Col

2

▼

RPT-nn7sss CALL US01 (result-v task-v)

- *Rpt-nn*, coded in columns 2 and 3, specifies a 2-digit number in the range 00 through 99 that identifies the CA Culprit report.
- *Sss*, coded in columns 5 through 7, specifies a 3-digit number indicating the sequence number of the type 7 parameter.
- **Result-v** (ARG1) requires the name of the field that receives the retrieved record.
- **Task-v** (ARG2) requires the name of a 1-character alphanumeric work field to hold user instructions and CULLUS01 return values:

The value...	Set by...	Means...
B	The user	Open the input file, access the first record, and return it to the receiving field.
S	The user	Close the input file.
E	CULLUS01	End-of-file and CULLUS01 has closed the file.
Blank	CULLUS01	Records were retrieved.

Fifth—**Test return values** in type 7 logic.

Sixth—**Define the external sequential file** in the CULP2 step of the CA Culprit job control language (JCL):

System	JCL statement
z/OS	//US01 DD DSN=user.inputfil,UNIT=tape,DISP=OLD, VOL=SER=nnnnnn
z/VM	Tape input: FILEDEF US01 TAP01 SL (OPTIONS Disk input: FILEDEF US01 DISK fn ft fm (OPTIONS ■ fn = file name of the external file ■ ft = file type of the external file ■ fm = file mode of the external file

Helpful hints

- If you use the CULLUS*nn* naming convention, you can prevent confusion by using numbers that are not found in the CA-supplied modules.
- An I/O error results in an abend. Check the system completion code that accompanies the abend to diagnose this error.
- The 1-character alphanumeric work field (*task-v*) first holds user instructions, which are later overwritten by CULLUS01 return values.
- When the return value is E, CULLUS01 has closed the file. An abend results if CULLUS01 is called again to close the file with a user-set value of S.

Example

This example uses CULLUS01 to retrieve customer account numbers from a sequential file during a database run.

The following code:

- Defines a database run
- Allows 1000 bytes (default) for the input buffer
- Uses a REC parameter (FIELD1) to define 5 bytes in the input buffer for the customer account number returned by US01
- Initializes work field EOFS to B, which directs CULLUS01 to open the input file and get the first record
- Issues a CALL to CULLUS01 from type 7 logic for record retrieval during the CA Culprit run
- Tests for end-of-file

```

DATABASE DICTNAME=DOCU DICT
INPUT DB SS=EMPSS01
PATHAA EMPLOYEE
REC FIELD1 95 5 $READ IN ACCOUNT NUMBER USING //US01 DD
010 E0FS 'B'
010 TEST ' '
013CULLUS01
0151*010 EMP-NAME-0415 HH 'NAME'
0151*020 FIELD1 HH 'ACCOUNT'
0151*040 TEST HH 'RETRIEVAL'
017010 CALL US01 (FIELD1,E0FS)
017 IF E0FS = ' ' 200
017 IF E0FS = 'E' STOP
017200 MOVE 'OK' TO TEST

```

REPORT NO. 01	NAME	CULLUS01 ACCOUNT	mm/dd/yy PAGE RETRIEVAL 1
	KATHERINE O'HEARN	15060	OK
	PHINEAS FINN	21056	OK
	NANCY TERNER	29557	OK
	BETH CLOUD	30115	OK
	JAMES JACOBI	33470	OK
	TOM FITZHUGH	69876	OK
	DOUGLAS KAHALLY	99083	OK

System Time and Date Retrieval (CULLUS10)

What You Can Do

CULLUS10 retrieves the system time and date. You can use CULLUS10 to retrieve the:

- Date in *mmddyy* format
- Year in *yy* format
- Month
- Day
- Day and time
- Time in *hhmmss* format
- Date in *mmddccyy* format
- Year in *ccyy* format

How to Use CULLUS10

To invoke CULLUS10:

First—**Define a 1-character alphanumeric work field**, initialized with a return format code (see below under *format-v*) or a blank, as the sending field.

Second—**Define an 8-byte numeric work field**, initialized to zero, for each receiving field.

Third—**Issue a CALL to CULLUS10** in type 7 logic:

Col

2

▼

RPT-*nn*7*sss*CALLUS*nn*(**format-v** **date-mdcy-v** **date-ccyy-v** **date-mm-v** **date-dd-v** **time-v**)

- *Rpt-nn*, coded in columns 2 and 3, specifies a 2-digit number in the range 00 through 99 that identifies the CA Culprit report.
- *Sss*, coded in columns 5 through 7, specifies a 3-digit number indicating the sequence number of the type 7 parameter.
- **Format-v** (ARG1) requires the name of the 1-character alphanumeric work field containing the return format code:

Format code...	Specifies...	And returns the...
'2'	mmddy	System date
'3'	yy	System year
'4'	mm	System month
'5'	dd	System day
'6'	mmddy/hhmmss	System date and time
'7'	hhmmss	System time
'8'	mmddccyy	System date with century
'9'	ccyy	System year with century
'0'	mmddccyy/hhmmss	System date and time with century

- **Date-mdcy-v** (ARG2) requires the name of the 8-byte numeric work field that receives the system date in *mmddyy* or *mmddccyy* format.
- **Date-ccyy-v** (ARG3) requires the name of the 8-byte numeric work field that receives the system year in *yy* or *ccyy* format.
- **Date-mm-v** (ARG4) requires the name of the 8-byte numeric work field that receives the system month.
- **Date-dd-v** (ARG5) requires the name of an 8-byte numeric work field that receives the system day.
- **Time-v** (ARG6) requires the address of an 8-byte numeric work field that receives the system time.

Fourth—**Test the value of the sending work field** in type 7 logic. If the value is 'E', an invalid field specification exists.

Helpful Hints

- To improve run-time efficiency, code your logic so that CULLUS10 is invoked only once.
- Instead of using CULLUS10 to print the system date on report headings, code the CA Culprit reserved word DATE on a type 4 parameter. The system date will print in the *mm/dd/yy* format.
- Use global work fields (GW0) when more than one report in the run requires system time or date output rather than calling CULLUS10 in each report.
- When testing for the value of the sending work field, branch to a type 7 statement that causes a dump (ZERO / ZERO ZERO) or calls CULLUS48 to issue a run-time message.
- Canadian users can specify DS=C on the PROFILE parameter and retrieve the system date in *yy/mm/dd* format on report titles.

Example 1

This example is a daily balance report that has the system date and time printed as subtitles.

The following code:

- Defines these work fields:
 - FORMAT, an alphanumeric work field, is assigned 6 as a format code.
 - WK-MDY, by default an 8-byte numeric work field, receives the system date retrieved by CULLUS10.
 - WK-TIME, by default an 8-byte numeric work field, receives the system time retrieved by CULLUS10.
- Uses a SORT/NOSORT parameter to make the current value of WK-MDY and WK-TIME available to the type 4 parameter references.
- Issues a CALL to CULLUS10 in type 7 logic to retrieve the date and time. Since ARG3 through ARG5 are not used, zeros act as position holders.

```
IN 80 F 80
REC NAME      5    25
REC BALANCE 160    7    3    DP=2
010 FORMAT '6'
010 WK-MDY
010 WK-TIME
013 CULLUS10
010OUT 60 D
01SORT WK-MDY WK-TIME NOSORT
01410001 ' '
01420001 'DATE:'
01420007 WK-MDY      FD
01430001 'TIME'
01430007 WK-TIME     FM '99.99.99'
01440001 ' '
01510001 NAME        HH 'NAME'
01510022 BALANCE     HH 'BALANCE'
017      CALL US10 (FORMAT WK-MDY 0 0 0 WK-TIME)
```

REPORT NO. 01 CULLUS10 mm/dd/yy PAGE 1

DATE: mm/dd/yy
TIME 17.32.56

	NAME	BALANCE
TERRY	JANENS E	38,000.00
JOE	NGUYA	31,000.00
MARK	TIME	33,000.00
ROGER	WILCO	80,000.00
ALBERT	BREEZE	38,000.00
CAROLYN	CROW	37,500.00
BURT	LANCHESTER	54,500.00
RENE	MAKER	85,000.00
MARYLOU	JOHNSON	12.00

Example 2

This example is a daily balance report that has the system date and time printed as subtitles. It is similar to Example 1, except the date is retrieved with the century.

The following code:

- Defines these work fields:
 - FORMAT, an alphanumeric work field, is assigned 0 as a format code.
 - WK-MDC, by default an 8-byte numeric work field, receives the system date retrieved by CULLUS10.
 - WK-TIME, by default an 8-byte numeric work field, receives the system time retrieved by CULLUS10.
- Uses a SORT/NOSORT parameter to make the current value of WK-MDC and WK-TIME available to the type 4 parameter references.
- Issues a CALL to CULLUS10 in type 7 logic to retrieve the date and time. Since ARG3 through ARG5 are not used, zeros act as position holders.

```

IN 80 F 80
REC NAME      5    25
REC BALANCE 160    7    3    DP=2
020 FORMAT '0'
020 WK-MDC
020 WK-TIME
023 CULLUS10
020UT 60 D
02SORT WK-MDC WK-TIME NOSORT
02410001 ' '
02420001 'DATE:'
02420007 WK-MDC      FM '99/99/9999'
02430001 'TIME'
02430007 WK-TIME     FM '99.99.99'
01440001 ' '
01510001 NAME        HH 'NAME'
01510022 BALANCE     HH 'BALANCE'
017      CALL US10 (FORMAT WK-MDC 0 0 0 WK-TIME)

```

REPORT NO. 02 CULLUS10 mm/dd/yy PAGE 1

DATE: mm/dd/yyyy

TIME 19.26.21

	NAME	BALANCE
TERRY	JANENS E	38,000.00
JOE	NGUYA	31,000.00
MARK	TIME	33,000.00
ROGER	WILCO	80,000.00
ALBERT	BREEZE	38,000.00
CAROLYN	CROW	37,500.00
BURT	LANCHESTER	54,500.00
RENE	MAKER	85,000.00
MARYLOU	JOHNSON	12.00

Julian Date Conversion (CULLUS11)

What You Can Do

You can use CULLUS11 to convert a Julian date (*yyddd*), stored as a zoned or packed decimal, to a packed decimal Gregorian date (*mmddyy*).

How to Use CULLUS11

To invoke CULLUS11:

1. **Define an 8-byte packed decimal numeric input field** or a **work field** to contain the Julian date if the input date is stored in zoned decimal format.

If the input field is stored as a zoned decimal:

- Move the input field to a numeric work field
- Use this work field as the sending field for the Julian date

2. **Define a numeric work field** to receive the Gregorian date.
3. **Test for invalid dates** in type 7 logic.
4. **Issue a CALL to CULLUS11** in type 7 logic:

Col

2

▼

RPT-nn7sss CALL US11 (jul-date-v greg-date-v)

- *Rpt-nn*, coded in columns 2 and 3, specifies a 2-digit number in the range 00 through 99 that identifies the CA Culprit report.
- *Sss*, coded in columns 5 through 7, specifies a 3-digit number indicating the sequence number of the type 7 parameter.
- **Jul-date-v** (ARG1) requires the name of the 8-byte packed decimal field containing the Julian date.
- **Greg-date-v** (ARG2) requires the name the numeric work field that receives the Gregorian date.

Helpful Hints

- Include error checking for invalid dates in the CA Culprit code. CULLUS11 does not issue an error message if the Julian day is less than 001 or greater than 366.
- CULLUS11 interprets a 00 year as a leap year. Dates in 1900 and 2100 will be interpreted incorrectly.

Example

This example uses CULLUS11 to convert an input date in Julian format (*yyddd*) to an output date in Gregorian format (*mmddy*).

The following code:

- Defines numeric work fields for the Julian input date and the Gregorian date
- Checks for invalid dates
- Moves the input Julian date to the numeric work field JUL-DATE
- Issues a CALL to CULLUS11

```
IN 80 F 400
REC JUL-IN-DATE 1 5 2
REC JUL-IN-DAY 3 3 2
990 JUL-DATE 0
990 GREG-DATE 0
993CULLUS11
9951*010 JUL-DATE FM '99.999' HH 'JULIAN DATE'
9951*020 GREG-DATE FD HH 'GREGORIAN DATE'
997100 JUL-IN-DAY LT 001 DROP
997150 JUL-IN-DAY GT 366 DROP
997200 MOVE JUL-IN-DATE JUL-DATE
997250 CALL US11 (JUL-DATE GREG-DATE)
```

REPORT NO. 99	CULLUS11	mm/dd/yy	PAGE	1
JULIAN DATE		GREGORIAN DATE		
96.299		10/25/96		
92.365		12/30/92		
90.100		04/10/90		
88.005		01/05/88		
94.050		02/19/94		
95.333		11/29/95		
91.083		03/24/91		

Century Date Conversion (CULLUS12)

CULLUS12 converts any century date to a user-specified format.

What You Can Do

You can use CULLUS12 to convert any date to any specified format. For example, you can:

- Convert Julian, Gregorian, European, and Canadian dates into any other specified format
- Use input dates that are stored as 8-byte alphanumeric or zoned decimal fields
- Use input dates that are stored as 8-byte packed decimal fields

How to Use CULLUS12

To invoke CULLUS12:

- First—**Define the input date:**
 - On a REC parameter, if the date is part of an input record
 - On a work field, if the value is supplied during the CULPRIT run
- Second—**Issue one or more calls to CULLUS12** in type 7 logic:

Col
2
▼
RPT-nn7sss CALL (input-field-v date-type-qv input-format-code-qv
output-format-code-qv output-field-v)
- *Rpt-nn*, coded in columns 2 and 3, specifies a 2-digit number in the range 00 through 99 that identifies the CULPRIT report.
- *Sss*, coded in columns 5 through 7, specifies a 3-digit number indicating the sequence number of the type 7 parameter.
- **Input-field-v** (ARG1) requires the name of the field containing the date.
- **Date-type-qv** (ARG2), enclosed in single quotation marks, requires an alphanumeric literal to define the data type of the input field (ARG1):

The literal...	The data type...
'Z'	8-byte alphanumeric or zoned decimal
'P'	8-byte packed decimal

- **Input-format-code-qv** (ARG3), enclosed in single quotation marks, requires the name of a 3-character alphanumeric work field or alphanumeric literal that defines the format of **input-field-v** (ARG1):

Format code	Input date format
'MDC'	mmddccyy
'MCD'	mmccyydd
'DMC'	ddmmccyy
'DCM'	ddccyymm
'CMD'	ccyyymmdd
'CDM'	ccyyddmm
'CDD'	ccyyddd

- **Output-format-code-qv** (ARG4), enclosed in single quotation marks, requires the name of a 3-character alphanumeric work field or alphanumeric literal that specifies the date format for **output-field-v** (ARG5). Valid format codes appear above.
- **Output-field-v** (ARG5) requires the name of an 8-byte packed decimal work field that receives the converted date.

Helpful Hints

- You can issue one or more calls to CULLUS12 from type 7 logic.
- CULLUS12 checks for invalid dates and values:
 - If an invalid date is encountered, the return date is converted to zero.
 - If an invalid data type or format code is encountered, the input date is converted to 999999.
- When using an output format code which requires a century (ccyy) and the input date does not include one, century will default to:
 - 20 if yy ≤ 40
 - 19 if yy > 40
- The year which determines the default century (40) can be overridden with the US12YR profile option when customizing the Culprit profile.

Note: For more information about this option, see the *CA Culprit for CA IDMS Reference Guide*.

Example

This example uses CULLUS12 to:

- Convert a zoned decimal input date in Canadian (*ccyyymmdd*) format to Gregorian format (*mmddyycc*)
- Convert a work field date in Canadian format (*ccyyymmdd*) to Julian format (*ccyyddd*)

The following code issues two calls to CULLUS12:

- The first CALL reads in a Canadian format zoned decimal input field, converts the date to a *mmddccyy* format, and stores the conversion in the work field WK-IN-DATE.
- The second CALL converts the work field WK-IN-DATE, converts the date to a *ccyyddd* format, and stores the conversion in the work field WK-JUL-DATE.

```
IN 80 F 400
REC IN-DATE 1 2
990 WK-IN-DATE
990 WK-JUL-DATE
993CULLUS12
99410001 ' '
9951*010 IN-DATE FM '9999/99/99' HH 'INPUT DATE'
9951*020 WK-IN-DATE FM '99/99/9999' HH 'CONVERTED' 'INPUT DATE'
9951*040 WK-JUL-DATE FM '9999.99' HH 'CONVERTED' 'JULIAN DATE'
997100 CALL US12 (IN-DATE 'Z' 'CDY' 'MDC' WK-IN-DATE)
997200 CALL US12 (WK-IN-DATE 'P' 'MDC' 'CDD' WK-JUL-DATE)
```

REPORT NO. 99	CULLUS12	mm/dd/yy PAGE 1	
	INPUT DATE	CONVERTED INPUT DATE	CONVERTED JULIAN DATE
	1994/02/09	02/09/1994	1994.040
	0093/12/25	12/25/1993	1993.359
	0001/12/25	12/25/2001	2001.359
	8888/88/88	00/00/0000	0000.000

Gregorian Date Conversion (CULLUS14)

What You Can Do

You can use CULLUS14 to convert a Gregorian date (*mmddyy*), stored as a zoned or packed decimal, to a packed decimal Julian date (*yyddd*).

How to Use CULLUS14

To invoke CULLUS14:

1. Define an 8-byte packed decimal numeric input field or a work field to contain the Gregorian date if the input date is stored in zoned decimal format.

If the input date is stored as a zoned decimal:

- Move the input field to a numeric work field
- Use the work field as the sending field for the Julian date

2. Define a numeric work field to receive the Julian date.

3. Issue a CALL to CULLUS14 in type 7 logic:

Col

2

▼

RPT-nn7sss CALL US14 (greg-date-v jul-date-v)

- Rpt-nn, coded in columns 2 and 3, specifies a 2-digit number in the range 00 through 99 that identifies the Advantage CA-Culpri report.
- Sss, coded in columns 5 through 7, specifies a 3-digit number indicating the sequence number of the type 7 parameter.
- Greg-date-v (ARG1) requires the name of an 8-byte packed decimal field containing the Gregorian date.
- Jul-date-v (ARG2) requires the name of a numeric work field that receives the Julian date.

Helpful Hint

CULLUS14 checks for month values in the range 01-12, day values in the range 01-31, and year values in the range 00-99.

Example 1—Gregorian to Julian format

This example shows the CA Culprit parameters required to convert a Gregorian (*mmddyy*) date to a Julian (*yyddd*) date.

The following code:

- Defines numeric work fields for Gregorian input date and the Julian date
 - Moves the Gregorian input date to the numeric work field GREG-DATE
 - Issues a CALL to CULLUS14
- ```
IN 80 F 400
REC GREG-IN-DATE 1 6 2
990 JUL-DATE 0
990 GREG-DATE 0
993CULLUS14
9951*010 GREG-DATE FD HH 'GREGORIAN DATE'
9951*020 JUL-DATE FM '99.999' HH 'JULIAN DATE'
997300 MOVE GREG-IN-DATE GREG-DATE
997350 CALL US14 (GREG-DATE JUL-DATE)
```

| REPORT NO. 99  | CULLUS14 | mm/dd/yy    | PAGE | 1 |
|----------------|----------|-------------|------|---|
| GREGORIAN DATE |          | JULIAN DATE |      |   |
| 01/10/90       |          | 90.010      |      |   |
| 10/21/56       |          | 56.295      |      |   |
| 12/25/87       |          | 87.359      |      |   |
| 03/01/85       |          | 85.060      |      |   |
| 03/17/94       |          | 94.076      |      |   |
| 09/17/86       |          | 86.260      |      |   |
| 09/08/93       |          | 93.251      |      |   |

Example 2—System to Julian format

This example shows the CA Culprit parameters required to retrieve the system date and convert it to a Julian date.

The following code uses two user modules:

- CULLUS10 to retrieve the system date
- CULLUS14 to convert the system date (*mmddyy*) to Julian format (*yyddd*)

Three work fields are required:

- CURR-DATE to receive code that specifies the format for the system date
- WORK-DATE to receive the system date from CULLUS10 and then send the date to CULLUS14
- JUL-DATE to receive the converted date

```
IN 200 F 4000
REC NAME 5 25
REC BALANCE 160 7 3 DP=2
013CULLUS10 AND CULLUS14
010 CURR-DATE '2'
010 WORK-DATE
010 JUL-DATE
010 ZERO
0150RT JUL-DATE NOSORT
010UT 80 D
01410001 ' '
01420001 'DATE:'
01420007 JUL-DATE FM '99.999'
01510001 NAME HH 'NAME'
01510032 BALANCE HH 'BALANCE'
017010 CALL US10 (CURR-DATE WORK-DATE 0 0 0 0)
017020 IF CURR-DATE EQ 'E' 120
017100 CALL US14 (WORK-DATE JUL-DATE)
017120 ZERO / ZERO ZERO
017150 STOP
```

| REPORT NO. 01 |            | CULLUS10 AND CULLUS14 | mm/dd/yy | PAGE | 1 |
|---------------|------------|-----------------------|----------|------|---|
| DATE: 99.278  |            |                       |          |      |   |
|               | NAME       | BALANCE               |          |      |   |
| TERRY         | JANENS E   | 38,000.00             |          |      |   |
| JOE           | NGUYA      | 31,000.00             |          |      |   |
| MARK          | TIME       | 33,000.00             |          |      |   |
| ROGER         | WILCO      | 80,000.00             |          |      |   |
| ALBERT        | BREEZE     | 38,000.00             |          |      |   |
| CAROLYN       | CROW       | 37,500.00             |          |      |   |
| BURT          | LANCHESTER | 54,500.00             |          |      |   |
| RENE          | MAKER      | 85,000.00             |          |      |   |

## Universal Date Conversion (CULLUS15)

CULLUS15 converts any date to a user-specified format.

### What You Can Do

You can use CULLUS15 to convert any date to any specified format. For example, you can:

- Convert Julian, Gregorian, European, and Canadian dates into any other specified format
- Use input dates that are stored as 6-byte alphanumeric or zoned decimal fields
- Use input dates that are stored as 8-byte packed decimal fields

### How to Use CULLUS15

To invoke CULLUS15:

First—**Define the input date:**

- On a REC parameter, if the date is part of an input record
- On a work field, if the value is supplied during the CA Culprit run

Second—**Issue one or more calls to CULLUS15** in type 7 logic:

Col

2

▼

RPT-nn7sss CALL US15 (input-field-v date-type-qv input-format-code-qv  
output-format-code-qv output-field-v)

- *Rpt-nn*, coded in columns 2 and 3, specifies a 2-digit number in the range 00 through 99 that identifies the CA Culprit report.
- *Sss*, coded in columns 5 through 7, specifies a 3-digit number indicating the sequence number of the type 7 parameter.
- **Input-field-v** (ARG1) requires the name of the field containing the date.
- **Date-type-qv** (ARG2), enclosed in single quotation marks, requires an alphanumeric literal to define the data type of the input field (ARG1):

| The literal... | The data type...                     |
|----------------|--------------------------------------|
| 'Z'            | 6-byte alphanumeric or zoned decimal |
| 'P'            | 8-byte packed decimal                |

| Format code | Input date format |
|-------------|-------------------|
| 'MDY'       | mmddyy            |
| 'MYD'       | mmyydd            |
| 'DMY'       | ddmmyy            |
| 'DYM'       | ddyymm            |
| 'YMD'       | yymmdd            |
| 'YDM'       | yyddmm            |
| 'YDD'       | yyddd             |

- **Input-format-code-qv** (ARG3), enclosed in single quotation marks, requires the name of a 3-character alphanumeric work field or alphanumeric literal that defines the format of **input-field-v** (ARG1):
- **Output-format-code-qv** (ARG4), enclosed in single quotation marks, requires the name of a 3-character alphanumeric work field or alphanumeric literal that specifies the date format for **output-field-v** (ARG5). Valid format codes appear above.
- **Output-field-v** (ARG5) requires the name of an 8-byte packed decimal work field that receives the converted date.

## Helpful Hints

- You can issue one or more calls to CULLUS15 from type 7 logic.
- CULLUS15 checks for invalid dates and values:
  - If an invalid date is encountered, the return date is converted to zero.
  - If an invalid data type or format code is encountered, the input date is converted to 999999.

### Example

This example uses CULLUS15 to:

- Convert a zoned decimal input date in Gregorian format (*mmddy*) to Canadian (*yymmdd*) format
- Convert a work field date in Julian format (*yyddd*) to Canadian format (*yymmdd*)

The following code issues two calls to CULLUS15:

- The first CALL reads in a Gregorian format zoned decimal input field, converts the date to a *yymmdd* format, and stores the conversion in the work field WK-IN-DATE.
- The second CALL reads the Julian date 86228 from the work field JUL-DATE, converts the date to a *yymmdd* format, and stores the conversion in the work field WK-JUL-DATE.

```

IN 80 F 400
REC IN-DATE 1 6 2
990 WK-IN-DATE
990 JUL-DATE.7 86228 86100 85030 83234 86341 85233 84078
990 WK-JUL-DATE
990 INDEX
993CULLUS15
99410001 ' '
9951*010 IN-DATE FD HH 'INPUT DATE'
9951*020 WK-IN-DATE FD HH 'CONVERTED' 'INPUT DATE'
9951*030 JUL-DATE.INDEX FM '99.999' HH 'JULIAN DATE'
9951*040 WK-JUL-DATE FD HH 'CONVERTED' 'JULIAN DATE'
997050 INDEX + 1 INDEX
997 INDEX GT 7 STOP
997100 CALL US15 (IN-DATE 'Z' 'MDY' 'YMD' WK-IN-DATE)
997200 CALL US15 (JUL-DATE.INDEX 'P' 'YDD' 'YMD' WK-JUL-DATE)

```

| REPORT NO. 99 | CULLUS15                |             | mm/dd/yy                 | PAGE | 1 |
|---------------|-------------------------|-------------|--------------------------|------|---|
| INPUT DATE    | CONVERTED<br>INPUT DATE | JULIAN DATE | CONVERTED<br>JULIAN DATE |      |   |
| 01/10/90      | 90/01/10                | 86.228      | 86/08/16                 |      |   |
| 10/21/56      | 56/10/21                | 90.100      | 90/04/10                 |      |   |
| 12/25/87      | 87/12/25                | 89.030      | 89/01/30                 |      |   |
| 03/01/85      | 85/03/01                | 93.234      | 93/08/22                 |      |   |
| 03/17/94      | 94/03/17                | 86.341      | 86/12/07                 |      |   |
| 09/17/86      | 86/09/17                | 95.233      | 95/08/21                 |      |   |
| 09/08/93      | 93/09/08                | 94.078      | 94/03/19                 |      |   |

## Random Access of ISAM Files (CULLUS22)

### What You Can Do

CULLUS22 retrieves specified ISAM file records during a standard CA Culprit run.

You can use CULLUS22 to:

- Retrieve records from an ISAM file by key
- Use more than one ISAM file in a single CA Culprit run
- Write all or part of the data from retrieved ISAM records to specific addresses

## How CULLUS22 Works

CULLUS22:

1. Uses a specified field from the input file as a retrieval key
2. Allows the retrieved ISAM records to be moved to an input field or a work field, as specified in the CA Culprit parameters
3. Uses a communication switch to:
  - Receive instructions to open or close the ISAM file
  - Return processing status information
4. Requires separately compiled and linked versions of CULLUS22 for each ISAM file accessed during a single CA Culprit run

## How to Use CULLUS22

### Preparing to Use CULLUS22

Define the ISAM data set in the execution JCL:

| System | Step | JCL statements *                                                                                                                                                                                                                                                                               |
|--------|------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| z/OS   | CULL | Define the ISAM file on the US22 DD statement:<br>//US22 DD DSN=user.file,UNIT=disk,<br>DISP=sbr,VOL=SER=nnnnnn                                                                                                                                                                                |
| z/VSE  | CULL | // ASSGN SYS004,DISK,VOL=nnnnnn<br>// DLBL filename,'file-id',ISE<br>// EXTENT SYS004,nnnnnn,4,1,rt,nt<br>// EXTENT SYS004,nnnnnn,1,2,rt,nt<br>// EXTENT SYS004,nnnnnn,2,3,rt,nt<br><br>rt = relative track where the<br>data extent begins<br>nt = number of tracks allocated<br>to this file |
| z/VM   |      | ISAM is not supported.                                                                                                                                                                                                                                                                         |

**Note:** \* See the *CA Culprit for CA IDMS Reference Guide* for a full explanation of the syntax.

If accessing more than one ISAM file:

First—**Modify the CULLUS22 Assembler source code:**

| System | Modification                                                    |
|--------|-----------------------------------------------------------------|
| z/OS   | Change the DDNAME keyword in the DCB macro                      |
| z/VSE  | Use unique filenames in the DTF macro for each copy of CULLUS22 |
| z/VM   | ISAM is not supported.                                          |

Second—**Reassemble and link edit each modification** of CULLUS22 using unique names.

### Using CULLUS22

To invoke CULLUS22:

First—**Define a field that contains the key** required to retrieve records from the ISAM file. The format of the key field depends on the operating system:

| System... | The key format...        |
|-----------|--------------------------|
| z/OS      | The same as the ISAM key |
| z/VSE     | 8-byte packed decimal    |

Second—**Define a dummy buffer area** equal to the length of the retrieved record.

- For non-database runs, use the MB= option on an additional INPUT parameter.
- For database runs, use one INPUT parameter that includes the extra storage requirement in the record size specification.

Third—Define a **1-byte alphanumeric work field** to act as a communications switch.

Fourth—**Issue a CALL to CULLUS22** from type 7 logic:

```
Col
2
```

▼

```
RPT-nn7sss CALLL US22 (return-position-v field-v key-v)
```

- *Rpt-nn*, coded in columns 2 and 3, specifies a 2-digit number in the range 00 through 99 that identifies the CA Culprit report.
- *Sss*, coded in columns 5 through 7, specifies a 3-digit number indicating the sequence number of the type 7 parameter.

- **Return-position-v** (ARG1) requires the name of an input or work field to which the retrieved ISAM record is moved.
- **Field-v** (ARG2) requires the name of a 1-byte work field that acts as a communications switch for passing information between type 7 code and CULLUS22. The work field can contain the following values:

| The value...  | Set by... | Means...                                                       |
|---------------|-----------|----------------------------------------------------------------|
| Any but C*    | User      | OPEN file and READ.                                            |
| C             | User      | CLOSE file.                                                    |
| Y             | CULLUS22  | Record found.                                                  |
| N             | CULLUS22  | Record not found.                                              |
| E             | CULLUS22  | Error on READ.                                                 |
| Z (z/OS ONLY) | CULLUS22  | OPEN file. Zero key length in the DCB.<br>(probable JCL error) |

**Note:** \* CULLUS22 accepts any value other than C as a READ instruction.

- **Key-v** (ARG3) requires the name of the ISAM key field used for retrieval. This argument can be set to an input or a work field. If set to a work field, the value will be stored in packed decimal format.

Fifth—**Test the value of the communications switch (ARG2)** in type 7 logic.

## Helpful Hints

- The key field is usually coded as an input field. If key values are to be altered during program execution, move values into a work field and use the work field as the key.  
If the generated key must be in a format other than packed decimal, a conversion can be performed by type 7 logic, a user-written module, or a CA-supplied procedure module.
- If multiple key fields are used, perform one of the following:
  - Combine keys into one field on the CALL US22 statement.
  - Modify CULLUS22 source code to accept additional arguments.

- If more than one ISAM file is used during a CA Culprit run, a separate version of CULLUS22 must be available for each file. In this case, assemble and link edit the module multiple times with different names.
- When multiple reports in a CA Culprit job use different records from a single ISAM file, invoke CULLUS22 for each report.
- When multiple reports in a CA Culprit job use the same record from a single ISAM file, invoke CULLUS22 once and pass a global work field to subsequent reports to indicate that the record was read successfully.

## Source Code Modifications

The CULLUS22 source code, shown below, must be modified when:

- Accessing multiple ISAM files in a z/OS environment
- Defining one or more ISAM files in a z/VSE environment

### z/OS Identifications

When more than one ISAM is to be accessed by CULLUS22:

1. Define the first ISAM file on the DD statement for US22 in CA Culprit JCL.
2. Define all other ISAM files by changing the DDNAME parameter in the DCB macro in CULLUS22 source code.
3. Reassemble and link edit the module, using a unique name such as CULLUS $nn$  where  $nn$  is a 2-digit number not used by any CA-supplied module.

### z/VSE Modifications

z/VSE users must modify CULLUS22 source code to define one or more ISAM files:

1. If the ISAM file key is not packed, change the CP instruction to CLC and modify the length of the move to match the length of the ISAM key.
2. If the length of the ISAM record is greater than 256 bytes, code additional MOVE statements, as appropriate.
3. If the ISAM key is not packed, change the ZAP instruction to MVC and modify the length of the instruction to match the ISAM key length.
4. Change the length and data type of LKEY to match the length of the ISAM key.

5. Change the length of the ISREC field to match the length of the ISAM file.
6. Modify the following parameters in the DTFIS macro:
  - DEVICE—Disk device for the ISAM file
  - DSKXTNT—EXTENTS specified at file creation time
  - HINDEX—Unit containing the highest index
  - KEYLEN—Number of bytes in the key
  - KEYLOC—Starting position of key on ISAM record
  - NRECDs—Number of records in a block
  - RECFORM—FIXUNB or FIXBLK
  - RECSIZE—ISAM record size
7. Modify the length of IOA1 to match the length of one ISAM block.
8. Assemble and link edit the module.

#### CULLUS22 Source Code

Source code modifications are required to define the ISAM files. In z/OS systems, modifications are required only when using multiple ISAM files. z/VSE requires modification to define all ISAM files used in the run.

```
CULLUS22 START 0
 USING CULLUS22,15
 B @START BRANCH AROUND HEAD MACRO
 CULHEAD NOCODE=YES,PATCH=NO
@START DS 0H
 STM 14,12,12(13) SAVE REGISTERS
 LR 12,15 LOAD BASE REGISTER
 DROP 15
 USING CULLUS22,R12 TELL THE ASSEMBLER ABOUT IT
 ST 13,SAVE+4 DO
 LR 10,13 STANDARD
 LA 13,SAVE SAVE AREA
 ST 13,8(10) LINKAGE
 LM 4,6,4(1) PICK UP ARGS 1 TO 3
 CLI 0(5),C'C' IS USER CALLING FOR CLOSE?
 BE CLOSEIT YES
```

|                                        |       |               |                                 |
|----------------------------------------|-------|---------------|---------------------------------|
| RESET                                  | NOP   | GETIT         | ACTIVE BRANCH ON ALL BUT 1ST    |
|                                        | MVI   | *-3,X'F0'     | SET BRANCH ON                   |
|                                        | OPEN  | ISFILE        | OPEN IS FILE                    |
| GETIT                                  | CLI   | LCODE,0       | IS IT 1ST TIME THRU?            |
|                                        | BE    | GETKEY        | YES - SKIP CLC W/LAST KEY       |
|                                        | CLI   | LCODE,C'E'    | WAS THERE I/O ERROR ON LAST GET |
|                                        | BE    | GETKEY        | YES - GO READ IT AGAIN          |
| ***                                    |       |               |                                 |
| ***                                    |       |               |                                 |
| CHANGE THE FOLLOWING INST AS REQUIRED  |       |               |                                 |
| (IF NOT PACKED DEC DO CLC ETC.)        |       |               |                                 |
| ***                                    |       |               |                                 |
|                                        | CP    | LKEY,0(8,6)   | LAST KEY VS. CURRENT REQUEST    |
|                                        | BE    | READOK        | SAME KEY -PROCESS AGAIN         |
| ***                                    |       |               |                                 |
| ***                                    |       |               |                                 |
| THE FOLLOWING CONVERTS P.DEC TO BINARY |       |               |                                 |
| REPLACE AS REQUIRED                    |       |               |                                 |
| ***                                    |       |               |                                 |
| GETKEY                                 | MVC   | DBLWD,0(6)    | MOVE KEY TO DOUBLE WORD         |
|                                        | CVB   | 7,DBLWD       | CONVERT TO BINARY (KEY FORMAT)  |
|                                        | ST    | 7,SKEY        | PUT IN KEY LOCATION             |
|                                        | READ  | ISFILE,KEY    | READ RECORD CALLED FOR BY KEY   |
|                                        | WAITF | ISFILE        | WAIT FOR I/O COMPLETION         |
|                                        | TM    | ISFILEC,X'CE' |                                 |
|                                        | BZ    | READOK        | NO I/O ERROR                    |
|                                        | MVI   | 0(5),C'E'     | SIGNAL I/O ERROR                |
|                                        | B     | NORECORD      | GO BLANK I/O AREA               |

```

READOK MVI 0(5),C'N' SET CODE TO NO RECORD
 TM ISFILEC,X'10'
 BO NORECORD NO
 MVI 0(5),C'Y' SET CODE TO RECORD FOUND

*** IF RECORD LENGTH EXCEEDS 256 BYTES 2 OR MORE
*** MVC STATEMENTS WOULD BE NEEDED

 MVC 0(L'ISREC,4),ISREC MOVE RECORD INTO USER AREA
 B RETURN GO BACK TO USER
NORECORD MVI 0(4),C' ' START BLANKING USER AREA

*** IF RECORD LENGTH EXCEEDS 256 BYTES 2 OR MORE
*** MVC STATEMENTS WOULD BE NEEDED

 MVC 1(L'ISREC-1,4),0(4) PROPAGATE BLANKS
 B RETURN GO BACK TO USER
CLOSEIT CLI LCODE,C'C' WAS LAST CALL TO CLOSE?
 BE RETURN GO BACK TO USER- ALREADY CLOSE
 CLOSE ISFILE CLOSE IS FILE AS REQUESTED
 MVI RESET+1,X'00' RESET SO IT WILL OPEN ON NXCALL
RETURN MVC LCODE,0(5) SAVE RETURN CODE

*** CHANGE TO MVC OR AS REQUIRED

 ZAP LKEY,0(8,6) SAVE KEY
 L 13,SAVE+4 RESTORE REG 13
 LM 14,12,12(13) RESTORE REGS 14 TO 12
 SR 15,15 CLEAR REG 15
 BR 14 RETURN TO USER
LCODE DC X'0' SAVE LAST RETURN CODE
DBLWD DS D DOUBLE WORD WORK AREA

*** SET LKEY TO LENGTH (& TYPE) OF INPUT KEY
*** SET SKEY TO LENGTH (& TYPE) OF ON FILE
*** SET ISREC TO LENGTH OF ONE RECORD (INCLUDING KEY)

LKEY DC PL8'0' SAVE LAST KEY HERE
SKEY DC F'0' CURRENT KEY REQUESTED

```

```

ISREC DS CL174 I/O WORK AREA (1 RECORD)
SAVE DS 18F LINKAGE SAVE AREA

*** SET DTFIS PARAMETERS AS REQUIRED

ISFILE DTFIS X
 DEVICE=2314, X
 DSKXTNT=6, X
 HINDEX=2314, X
 IOAREAR=IOA1, X
 IOROUT=RETRVE, X
 KEYARG=SKEY, X
 KEYLEN=4, X
 KEYLOC=1, X
 MSTIND=YES, X
 NRECD=4, X
 RECFORM=FIXBLK, X
 RECSIZE=174, X
 TYPEFLE=RANDOM, X
 WORKR=ISREC

*** SET IOA1 TO SIZE OF ONE BLOCK

IOA1 DS CL696 I/O AREA FOR DTFIS
 END

```

### Example

This sample code shows the CA Culprit parameters required to retrieve an ISAM record.

The following code:

- Defines the input file that contains a key field
- Defines a dummy buffer area, using the MB=D option of the INPUT parameter
- Defines fields for the retrieved ISAM records
- Defines the key field of the input file (KEY-FIELD) and the ISAM record (REC1) in the dummy buffer
- Defines a 1-byte work field (COMM-SW) to pass OPEN and CLOSE file instructions to CULLUS22 and return file status codes.
- Calls CULLUS22 to first open the ISAM file and then to close the file when no further records are found or when an error occurs
- Tests the file status codes of COMM-SW and continues processing, stops processing, or closes the file, depending on the value found

```
IN 80 F 400
REC KEY-FIELD 1 15 3
IN 80 F 80 MB=D
REC REC1 1 80 $START OF DUMMY BUFFER
REC FLD1 1 15 $FIELD 1 OF THE ISAM RECORD
REC FLD2 16 30 $FIELD 2 OF THE ISAM RECORD
010 COMM-SW 'Z'
0151*010 KEY-FIELD
017005 IF EOF EQ 200
017010 CALL US22 (REC1 COMM-SW KEY-FIELD)
017020 IF COMM-SW EQ 'Y' TAKE
017030 IF COMM-SW EQ 'E' 300
017200 MOVE 'C' TO COMM-SW
017010 CALL US22 (REC1 COMM-SW KEY-FIELD)
017 STOP
017300 $ERROR ROUTINE HERE
```

## Random Access of VSAM Files (CULLUS25)

### What You Can Do

CULLUS25 retrieves data from VSAM files during a standard CA Culprit run.

You can use CULLUS25 to:

- Access one or more key or entry-sequenced VSAM files during a CA Culprit run
- Retrieve all or part of a VSAM record by:
  - A full or partial key value
  - An exact or comparative (equal to or greater than) key value
- Write retrieved VSAM records to a work field or a dummy area in the CA Culprit input buffer

## How to Use CULLUS25

To invoke CULLUS25:

First—**Define the input file containing the retrieval key** on the INPUT parameter and in the CULL step of the CA Culprit JCL.

Second—**Define the VSAM file** on an INPUT parameter and in the CA Culprit JCL:

- For non-database runs, use the MB= DUMMY option on an additional INPUT parameter.
- For database runs, use one INPUT parameter that includes the extra storage requirement in the record size specification.
- Use the external filename SYS020 in CA Culprit JCL.

Third—**Define the key fields in each file** on REC parameters.

Fourth—**Issue a CALL to CULLUS25** in type 7 logic:

Col

2

▼

RPT-*nn*sss **CALL US25** (**conversion-qv search-v key-v length-vn return-v**  
**external-file-name-v var-length-field-v**)

- *Rpt-nn*, coded in columns 2 and 3, specifies a 2-digit number in the range 00 through 99 that identifies the CA Culprit report.
- *Sss*, coded in columns 5 through 7, specifies a 3-digit number indicating the sequence number of the type 7 parameter.
- **Conversion-qv** (ARG1) requires a 1-byte alphanumeric code, enclosed in single quotation marks, to specify the conversion of the unsigned bit string in the VSAM key field:

| The code... | Converts to...                                     |
|-------------|----------------------------------------------------|
| Blank       | None                                               |
| '1'         | Binary                                             |
| '2'         | Zoned decimal without changing the sign bit value  |
| '3'         | Packed decimal without changing the sign bit value |
| 'p'         | Packed decimal with the sign X'F'                  |
| 'U'         | Zoned decimal with the sign X'F'                   |
| '9'         | None. Closes the VSAM file                         |

- **Search-v** (ARG2) requires a numeric literal or the name of an optional 8-byte numeric field. Values of this field control the type of search and compare method used to access the VSAM file:

| Search value                                     | Search type    | Compare method                     | Record retrieved                                                                                  |
|--------------------------------------------------|----------------|------------------------------------|---------------------------------------------------------------------------------------------------|
| Zero                                             | Full key (FKS) | Key equal (KEQ)                    | The record with an identical key                                                                  |
| Less than the VSAM key field size                | Generic (GEN)  | Key greater than or equal to (KGE) | The first having <i>n</i> positions greater than or equal to <i>n</i> positions of the search key |
| Greater than or equal to the VSAM key field size | Full key (FKS) | Key greater than or equal to (KGE) | The first having a full key value greater than or equal to the search key                         |

- **Key-v** (ARG3) requires the name of the key field used for retrieving records from the VSAM file. If *conversion-qv* is nonblank, this field must be a work field containing the key value.
- **Length-vn** (ARG4) requires a numeric literal or the name of an 8-byte numeric field that contains the length of the receiving field (*return-v*).
- **Return-v** (ARG5) requires the name of a work field or a dummy buffer area field to receive the retrieved VSAM record.

When a NO RECORD FOUND condition occurs, CULLUS25 returns two asterisks (\*\*) followed by blank spaces to **return-v**.

- **External-file-name-v** (ARG6) requires an alphanumeric literal or the name of an 8-byte alphanumeric field that specifies the ddname, filename, or filedef of the VSAM file accessed. Ddname (z/OS) must be eight characters and blank filled to the right.

If **external-file-name-v** is blank or not used, CULLUS25 assumes an external file name of SYS020 for the file.

- **Var-length-field-v** (ARG7) requires the name of an 8-byte packed decimal work field that holds the current record length contained in a variable-length VSAM file.

Fifth—**Test for a NO RECORD FOUND condition** in type 7 logic.

## Helpful Hints

- To avoid overwriting *external-file-name-v* by another user module, specify the external file name (ARG6) each time the call to CULLUS25 is issued. Reset the value before the CALL is issued.
- Failure to close a VSAM file may result in the loss of the file.
- More than one VSAM file can be accessed in a single CA Culprit run by creating a copy of CULLUS25 for each file and changing the name to be unique (CULLUS26, CULLUS27,...). Either copy the module and rename it or linkedit the new version using the CULLUS25 load module.
- VSAM treats all key fields as unsigned binary strings. When testing a key field, use logical rather than arithmetic tests.
- If a key field is not a binary string, specify a data conversion in the first argument (**conversion-qv**) of the CALL.
- If the retrieved VSAM record exceeds the defined receiving field length, the record is truncated to the length of the receiving field (ARG5) when read into the input buffer.
- If the length of the receiving field is longer than the retrieved record, the unused portion of the field is not initialized to blanks. The contents are unpredictable.

### Example

This example shows the CA Culprit parameters required to retrieve VSAM records that have key fields corresponding to keys contained in an input file.

The following code:

- Uses INPUT parameters to define an input file containing a key field and a dummy input buffer area to receive the VSAM record.  
  
The INPUT parameter allocates 400 bytes for the dummy buffer. Since the receiving area is defined on the CALL statement as 400 bytes, only the first 400 bytes of the VSAM record is read into the input buffer.
- Uses REC parameters to define the key field of the input record and the key field of the VSAM record.
- Tests for an end-of-file condition. When end-of-file is reached, the logic branches to statement 200, which issues a CALL to CULLUS22 to close the file (ARG1).
- Issues a CALL to CULLUS25.
- Tests for a NO RECORD FOUND condition by looking for asterisks (\*\*) in the first two positions of KEY-FIELD2.
- Tests key field values. If the values are not equal, the VSAM record is dropped.

```
INPUT 80 F 4000
REC KEY-FIELD 1 2
INPUT 400 F 400 MB=D
REC KEY-FIELD2 1 2
.
.
.
017005 EOF EQ 200
017010 CALL US25 (' ' 0 KEY-FIELD 400 KEY-FIELD2)
017060 IF KEY-FIELD2 EQ '**' DROP
017070 IF KEY-FIELD NE KEY-FIELD2 DROP
017080 TAKE
017200 CALL US25 ('9')
017205 DROP
```

## Creating a Vertical Hexadecimal Dump (CULLUS29)

### What You Can Do

CULLUS29 creates a hexadecimal dump of a record, a portion of a record or field, or a work field without dumping the entire buffer area.

The vertical dump format consists of four printed lines:

```
Line 1 -- Character representation
Line 2 -- Zone representation
Line 3 -- Digit representation
Line 4 -- Scale
```

## How to Use CULLUS29

To invoke CULLUS29:

First—**Define the input or work field that holds the data** to be dumped.

Second—**Define a subscripted work field of 528 bytes** as the receiving field for the data to be dumped.

Third—**Divide the subscripted work field into 4 elements of 132 bytes** to represent the printed lines of the dump.

Fourth—**Define a 1-byte alphanumeric work field** to hold the status flag for the dump.

Fifth—**Issue a CALL to CULLUS29** in type 7 logic:

Col

2

▼

RPT-nn7sss CALL US29 (dump-field-name length-vn result-v status-v)

- *Rpt-nn*, coded in columns 2 and 3, specifies a 2-digit number in the range 00 through 99 that identifies the CA Culprit report.
- *Sss*, coded in columns 5 through 7, specifies a 3-digit number indicating the sequence number of the type 7 parameter.
- **Dump-field-name** (ARG1) requires the name of the dumped input or work field. The field can be in any format.
- **Length-vn** (ARG2) requires a numeric literal or an 8-byte packed decimal field to specify the length of the dumped record or field.
- **Result-v** (ARG3) requires the name of the subscripted work field receiving the dumped field.
- **Status-v** (ARG4) requires the name of a 1-byte alphanumeric work field that holds a status code for the dump:

| Code... | Means...                                                                            |
|---------|-------------------------------------------------------------------------------------|
| 'F'     | Initializes the field.                                                              |
| ' '     | The routine has not finished dumping the requested area.                            |
| 'E'     | The routine has completed generating the dump. No further lines should be released. |

Sixth—**Test the value of the status flag** work field immediately after the CALL to CULLUS29. If the status equals ' ', release the print lines and branch back to the CALL statement. Continue branching back until the value of the status flag equals 'E'.

## Helpful Hints

- The receiving field for the dump is a 528-byte subscripted work field. Although each type 5 line requires 132 characters, only 100 bytes of the dump prints.
- Reset the status flag (ARG4) to 'F' to reinvoke CULLUS29 after a previous dump in the same CA Culprit program has been completed.
- Always test the value of the status flag (ARG4) when control returns from CULLUS29 to the CA Culprit code.

### Example

This example uses CULLUS29 to create a dump of the first 50 bytes of a record.

The following code:

- Defines IN-FLD as the input field where the dump will start
- Defines work fields to receive:
  - The length of the input record to be dumped
  - The status code for the dump
  - The formatted dump lines
- Calls CULLUS29 from type 7 logic
- Tests the value of FLAG to determine if the dump is completed
- Repeats processing until CULLUS29 sets the value of FLAG to 'E', which indicates that the dump is completed

```

INPUT 200 F 200
REC IN-FLD 1 200
130 FLAG ' '
130 LENGTH 50
130 OUT0.528 ' '
133 CULLUS29
135100010 OUT0.1 SZ=132
13520001 OUT0.133 SZ=132
13530001 OUT0.265 SZ=132
13540001 OUT0.397 SZ=132
137600 MOVE 'F' TO FLAG
137620 CALL US29 (IN-FLD LENGTH OUT0.1 FLAG)
137 IF FLAG NE ' ' DROP
137 RELS (1 2 3 4)
137625 B 620

```

|                                                 |                  |                     |
|-------------------------------------------------|------------------|---------------------|
| REPORT NO. 13                                   | CULLUS29         | mm/dd/yy PAGE 1     |
| CHAR 047ITERRY                                  | JANENS E         | SESE-SEK0 E         |
| ZONE FFCECDDE44444D                             | CDCDE4C4444444   | ECEC6ECDD444444444C |
| NUMR 04793599800001                             | 1555205000000025 | 250252600000000005  |
| 01...5...10...5...20...5...30...5...40...5...50 |                  |                     |

## Obtaining Hexadecimal Representation (CULLUS31)

### What You Can Do

You can use CULLUS31 to translate an input or work field into the hexadecimal representation of a 1- to 25-byte field that is stored in any format.

### How to Use CULLUS31

To invoke CULLUS31:

1. **Define the field to be translated** on a REC parameter or a work field parameter.
2. **Define an alphanumeric work field** to receive the translated value.
3. **Issue a CALL to CULLUS31** from type 7 logic:

Col

2

▼

RPT-*nn*sss **CALL US31** (**field-name length-vn result-v**)

- *Rpt-nn*, coded in columns 2 and 3, specifies a 2-digit number in the range 00 through 99 that identifies the CA Culprit report.
- *Sss*, coded in columns 5 through 7, specifies a 3-digit number indicating the sequence number of the type 7 parameter.
- **Field-name** (ARG1) requires the name of the input or work field passed to CULLUS31 for translation.
- **Length-vn** (ARG2) requires a numeric literal or the name of an 8-byte packed decimal field whose value specifies the length of the field being translated (*field-name*).
- **Result-v** (ARG3) requires the name of the receiving field for the hexadecimal representation of *field-name*. The length of this field must be twice as long as the field being translated.

### Example

This example shows the CA Culprit parameters required to translate the contents of an input field into a hexadecimal representation.

The following code:

- Uses a REC parameter to define an input field (IN-FLD) as a 4-character alphanumeric input field
- Uses a work field parameter to define the output field (OUT-FLD) as a 12-byte receiving field for the hexadecimal translation
- Issues a CALL to CULLUS31 using a numeric literal (4) to specify the length of the input field

```
INPUT 200 F 200
REC IN-FLD 1 4
130 OUT-FLD '12345678'
133 CULLUS31
1351*001 IN-FLD
1351*005 OUT-FLD
137100 CALL US31 (IN-FLD 4 OUT-FLD)
```

| REPORT NO. 13 | CULLUS31           | mm/dd/yy | PAGE | 1 |
|---------------|--------------------|----------|------|---|
| INPUT         | HEX REPRESENTATION |          |      |   |
| 047I          | F0F4F7C9           |          |      |   |
| 046I          | F0F4F6C9           |          |      |   |
| 035E          | F0F3F5C5           |          |      |   |
| 034I          | F0F3F4C9           |          |      |   |
| 046G          | F0F4F6C7           |          |      |   |
| 033D          | F0F3F3C4           |          |      |   |
| 030A          | F0F3F0C1           |          |      |   |
| 001E          | F0F0F1C5           |          |      |   |
| 045H          | F0F4F5C8           |          |      |   |
| 006I          | F0F0F6C9           |          |      |   |

## Converting Packed Decimal to Binary (CULLUS33)

### What You Can Do

You can use CULLUS33 to translate packed decimal values in the range of -2,147,483,647 to +2,147,483,647 (231 - 1) into a binary format.

CULLUS33 is particularly useful in creating a record descriptor word (RDW) that is used in the first four bytes of variable-length records.

## How to Use CULLUS33

To invoke CULLUS33:

1. **Define the packed field** that contains the value to be converted.
2. **Define a dummy buffer area or alphanumeric work field** to receive the binary conversion.
  - For non-database runs, use the MB= option on an additional INPUT parameter.
  - For database runs, use one INPUT parameter that includes the extra storage requirement in the record size specification.
3. **Issue a CALL to CULLUS33** from type 7 logic:

Col  
2

▼

RPT-nn7sss CALL US33 (field-name input-length-vn result-v result-length-vn)

- *Rpt-nn*, coded in columns 2 and 3, specifies a 2-digit number in the range 00 through 99 that identifies the CA Culprit report.
- *Sss*, coded in columns 5 through 7, specifies a 3-digit number indicating the sequence number of the type 7 parameter.
- **Field-name** (ARG1) requires the name of the packed decimal input or work field to be converted.
- **Input-length-vn** (ARG2) requires a numeric literal or an 8-byte packed decimal work field specifying the length of the field to be converted.
- **Result-v** (ARG3) requires the name of the field receiving the binary conversion.
- **Result-length-vn** (ARG4) requires a numeric literal or 8-byte packed decimal work field specifying the length of the receiving field **result-v**. The size of the result field must be large enough to accommodate the value of the field being converted:

| The decimal value...             | Results in the field size... |
|----------------------------------|------------------------------|
| 0 - 127                          | 1 byte                       |
| -32,767 to +32,767               | 2 bytes                      |
| 0 to +8,388,607                  | 3 bytes                      |
| -2,147,483,647 to +2,147,483,647 | 4 bytes                      |

## Helpful Hints

- A binary output field can also be created with the FB format code placed on a type 5 parameter.
- Inaccurate specification of the input field length produces unpredictable binary results.
- High order bytes are truncated if the size of the receiving field is too small.
- If CULLUS33 is invoked to create the RDW for variable-length output records, the receiving field should be defined as alphanumeric. If the result is placed in a binary field, CA Culprit attempts to output the number in zoned decimal format.

### Example

This example shows the CA Culprit parameters required to convert a packed decimal input field to binary output.

The following code:

- Defines the input file containing the field to be converted (PACKED-NUMBER)
- Defines a dummy input buffer area (BINARY-RESULT) to receive the conversion
- Issues a CALL to CULLUS33 from type 7 logic

```
INPUT 80 F 4000
REC PACKED-NUMBER 1 8 3
INPUT 80 F 80 MB=D
REC BINARY-RESULT 1 4
.
.
.
997110 CALL US33 (PACKED-NUMBER 8 BINARY-RESULT 4)
```

## Converting Packed Decimal to Zoned Decimal (CULLUS34)

### What You Can Do

You can use CULLUS34 to convert a packed decimal field to zoned decimal or alphanumeric format.

## How to Use CULLUS34

To invoke CULLUS34:

1. **Define the input file or work field** that contains the packed decimal field to be converted.
2. **Define a dummy area** that allocates space for the receiving field.
3. **Issue a CALL to CULLUS34** from type 7 logic:

```
Col
2
```

▼

```
RPT-nn7sss CALL US34 (field-name result-v input-length-vn)
```

- *Rpt-nn*, coded in columns 2 and 3, specifies a 2-digit number in the range 00 through 99 that identifies the CA Culprit report.
- *Sss*, coded in columns 5 through 7, specifies a 3-digit number indicating the sequence number of the type 7 parameter.

- **Field-name** (ARG1) requires the name of the 1- to 16-byte packed decimal input or work field.
- **Result-v** (ARG2) requires the name of an alphanumeric result field that is one byte less than twice the length of the input field (*field-name*).

The hexadecimal representation of each digit is in the range X'F0' through X'F9'. Negative values contain the negative sign in the zoned position of the last byte. Positive values contain an F in the zoned position of the last byte.

- **Input-length-vn** (ARG3) requires a numeric literal or the name of an 8-byte packed decimal work field that specifies the length of the packed decimal input field (*field-name*). The range is 1 through 16.

If the length of the input field exceeds 16, no conversion occurs.

### Example

This example converts numbers that are stored in packed decimal format to zoned decimals. Receiving fields hold the first five, next three, and last five digits.

The following code:

- Defines the input file containing the packed decimal field PKD-NUM
- Defines a dummy input buffer area to receive the conversion
- Issues a CALL to CULLUS34 from type 7 logic to perform the conversion

```
IN 200 F 400
REC PKD-NUM 160 7 3
IN 13 MB=DUMMY
REC NUMBER 1 13
REC FIRST-FIVE 1 5
REC NEXT-THREE 6 3
REC LAST-FIVE 9 5
010UT 80 D
013CULLUS34
01410001 ' '
0151*002 NUMBER FN HH 'THE NUMBER'
0151*003 NEXT-THREE HH 'MIDDLE THREE' 'DIGITS'
017001 CALL US34 (PKD-NUM NUMBER 7)
```

|               |                     |                 |
|---------------|---------------------|-----------------|
| REPORT NO. 01 | CULLUS34            | mm/dd/yy PAGE 1 |
| THE NUMBER    | MIDDLE THREE DIGITS |                 |
| 0000003800000 | 038                 |                 |
| 0000003100000 | 031                 |                 |
| 0000003300000 | 033                 |                 |
| 0000008000000 | 080                 |                 |
| 0000003800000 | 038                 |                 |
| 0000003750000 | 037                 |                 |
| 0000005450000 | 054                 |                 |

## Interpreting Bit Settings (CULLUS35)

### What You Can Do

You can use CULLUS35 to interpret the bit settings for a string of bytes.

### How to Use CULLUS35

To invoke CULLUS35:

First—**Define the input or work field** to be interpreted.

Second—**Issue a CALL to CULLUS35** from type 7 logic:

```
Col
2
▼
RPT-nn7sss CALL US35 (field-name length-vn result-v
representation-length-n start-position-a bit-count-n)
```

- *Rpt-nn*, coded in columns 2 and 3, specifies a 2-digit number in the range 00 through 99 that identifies the CA Culprit report.
- *Sss*, coded in columns 5 through 7, specifies a 3-digit number indicating the sequence number of the type 7 parameter.
- **Field-name** (ARG1) requires the name of the input or work field to be interpreted.
- **Length-vn** (ARG2) requires a numeric literal or the name of an 8-byte packed decimal work field specifying the length of the field being interpreted (*field-name*).
- **Result-v** (ARG3) requires the name of a receiving field large enough to accommodate the bit settings for the entire input field. The length of *result-v* depends on whether 2 or 3 positions are used for bit representation, as described below in *representation-length-n*.
- **Representation-length-n** (ARG4) requires the name of a field containing a value that specifies whether 2 or 3 positions are used for each bit representation:

| Bit representation    | Result field size |
|-----------------------|-------------------|
| 2-positions (00,01)   | 23 X length-vn    |
| 3-positions (000,001) | 31 X length-vn    |

- **Start-position-a** (ARG5) requires the name of a field that specifies the byte position where bit conversion begins. *Start-position-a* applies only to fields having multiple bytes. The default is 0.
- *Bit-count-n* (ARG6) is an optional argument using 0 or 1 as a numeric literal or the name of an 8-byte packed decimal field containing a 0 or 1 to specify the manner in which bits are counted. The default is 0.

0 counts bits as 00 01 02 03 04 05 06 07.

1 counts bits as 01 02 03 04 05 06 07 08.

## Helpful Hints

CULLUS35 displays bit settings by listing the positions of the ON bits.

For example, the bit settings shown below are represented by the display 01,03,04,05.

| 1st Bit | 2nd Bit | 3rd Bit | 4th Bit | 5th Bit | 6th Bit | 7th Bit | 8th Bit |
|---------|---------|---------|---------|---------|---------|---------|---------|
| (00)    | (01)    | (02)    | (03)    | (04)    | (05)    | (06)    | (07)    |
| OFF     | ON      | OFF     | ON      | ON      | ON      | OFF     | OFF     |

### Example

This example interprets the bit settings of an asterisk (\*).

The following code:

- Defines 23-byte alphanumeric work fields (RESULT0, RESULT1) to receive the bit settings
- Defines a work field (FIELD) to contain the hexadecimal value X'F8'
- Issues calls to CULLUS35 from type 7 logic to interpret the bit settings in each bit-count format

```
IN 80 F 80
REC ASTERISK 1 1
020 RESULT0 '
020 RESULT1 '
020 FIELD X'F8'
023CULLUS35
0251*005 FIELD HH 'FIELD'
0251*010 RESULT0 HH 'BINARY REPRESENTATION' 'USING BINARY COUNT 0'
0251*015 RESULT1 HH 'BINARY REPRESENTATION' 'USING BINARY COUNT 1'
027010 CALL US35 (FIELD 1 RESULT0 2 0)
027020 CALL US35 (FIELD 1 RESULT1 2 0 1)
```

| REPORT NO. | 02                                            | CULLUS35                                      | mm/dd/yy | PAGE | 1 |
|------------|-----------------------------------------------|-----------------------------------------------|----------|------|---|
| FIELD      | BINARY REPRESENTATION<br>USING BINARY COUNT 0 | BINARY REPRESENTATION<br>USING BINARY COUNT 1 |          |      |   |
| 8          | 00,01,02,03,04                                | 01,02,03,04,05                                |          |      |   |

## Converting Floating Point Values to Packed Decimal(CULLUS36)

### What You Can Do

You can use CULLUS36 to convert a single or double floating point value to a 16-byte packed decimal format.

## How to Use CULLUS36

To invoke CULLUS36:

First—**Define the input file and the field containing the floating point values.**

Second—**Define work fields** to receive the converted value (16-byte numeric field) and hold the precision indicator (1-byte alphanumeric field) if an alphanumeric literal is not used.

Third—**Issue a CALL to CULLUS36** in type 7 logic:

Col

2

▼

RPT-*nn*sss CALL US36 (**field-name indicator-qv result-v**  
**decimal-place-vn end-code-v**)

- *Rpt-nn*, coded in columns 2 and 3, specifies a 2-digit number in the range 00 through 99 that identifies the CA Culprit report.
- *Sss*, coded in columns 5 through 7, specifies a 3-digit number indicating the sequence number of the type 7 parameter.
- **Field-name** (ARG1) requires the name of the input field that contains the single or double precision floating point value.
- **Indicator-qv** (ARG2) is a 1-byte literal, enclosed in single quotation marks, or the name of an alphanumeric work field that contains the precision indicator:

| Precision... | Is indicated by... |
|--------------|--------------------|
| Single       | 'S'                |
| Double       | 'D'                |

- **Result-v** (ARG3) requires the name of a 16-byte packed decimal work field to receive the converted value.
- **Decimal-place-vn** (ARG4) requires an integer or the name of an 8-byte packed decimal work field that specifies the number of decimal places returned in the output. The range is 0 through 14.
- **End-code-v** (ARG5) requires the name of a 4-byte alphanumeric work field whose value specifies whether the conversion is successful:

| The value... | Indicates the conversion is...         |
|--------------|----------------------------------------|
| Blank        | Successful                             |
| ARGn         | Unsuccessful in the argument indicated |

Fourth—**Test the value of the error indicator work field** (*end-code-v*) immediately after the US36 call.

## Helpful Hints

- The COBOL equivalents for floating point values and the corresponding field lengths are:

| Precision | COBOL equivalent | Field length |
|-----------|------------------|--------------|
| Single    | COMP-1           | 4 bytes      |
| Double    | COMP-2           | 8 bytes      |

- The number of decimal places specified for output (*decimal-place-vn*) should agree with the DP= specification of the work field that receives the converted value (*result-v*).
- If the result overflows the space allocated:
  - The rightmost digits are truncated to the right of the decimal point.
  - The leftmost digits are truncated to the left of the decimal point.

### Example

This example shows the CA Culprit parameters required to convert single- and double-precision values to packed decimal format.

The following code:

- Defines the single- (FIELD1) and double- (FIELD2) precision fields on REC parameters
- Defines work fields for the result fields (PACKED-SINGLE and PACKED-DOUBLE) and an error code field for each conversion (CODE-1 and CODE-2)
- Issues separate calls to CULLUS36 to convert the single-precision field and then the double-precision field
- Tests for successful conversion after each call. If the conversion is not successful, an error handling routine executes.

```

INPUT 80 F 2960
REC FIELD1 1 4
REC FIELD2 5 8
990 PACKED-SINGLE DP=2
990 PACKED-DOUBLE DP=3
990 CODE-1 '1234'
990 CODE-2 ' ' ' '
.
.
.
997010 CALL US36 (FIELD1 'S' PACKED-SINGLE 2 CODE-1)
997015 CODE-1 NE ' ' ' 200
997020 CALL US36 (FIELD2 'D' PACKED-DOUBLE 3 CODE-2)
997025 CODE-2 NE ' ' ' 200
997100 TAKE
997200 $ERROR HANDLING ROUTINE IS CODED HERE

```

## Converting Doubleword Binary to Packed Decimal (CULLUS37)

### What You Can Do

You can use CULLUS37 to convert a doubleword binary field into an 8-byte packed decimal format. This conversion is particularly useful in converting binary input that exceeds the REC parameter's limit (4 bytes) to an 8-byte packed decimal work field.

## How to Use CULLUS37

To invoke CULLUS37:

1. **Define the binary input field as alphanumeric** on a REC parameter.
2. **Define an 8-byte numeric work field.**
3. **Issue a CALL to CULLUS37** in type 7 logic:

Col

2

▼

RPT-nn7sss CALL US37 (**input-field-v** **result-field-v**)

- *Rpt-nn*, coded in columns 2 and 3, specifies a 2-digit number in the range 00 through 99 that identifies the CA Culprit report.
- *Sss*, coded in columns 5 through 7, specifies a 3-digit number indicating the sequence number of the type 7 parameter.
- **Input-field-v** (ARG1) requires the name of the 8-byte binary input field.
- **Result-field-v** (ARG2) requires the name of the work field receiving the packed decimal conversion.

### Example

This example shows the CA Culprit parameters required to convert an 8-byte binary input field to packed decimal format.

- The binary field DOUBLE-BIN is defined as alphanumeric on the REC parameter.
- The work field that receives the converted value is defined.
- A CALL to CULLUS37 is issued in type 7 logic.

INPUT 80 F 4000

REC DOUBLE-BIN 1 8

010 WORK-FIELD

.

.

.

017001 CALL US37 (DOUBLE-BIN WORK-FIELD)

## Sending Messages (CULLUS40)

### What You Can Do

z/VSE users can use CULLUS40 to send messages to and receive messages from the console operator.

## How to Use CULLUS40

To invoke CULLUS40:

1. **Define a 25-byte alphanumeric work field** to hold the message sent.
2. **Define another 25-byte alphanumeric work field** to receive the message returned.
3. **Issue a CALL to CULLUS40** from type 7 logic:

Col

2

▼

RPT-*nn**sss* CALL US40 (*message-qv* *response-v*)

- *Rpt-nn*, coded in columns 2 and 3, specifies a 2-digit number in the range 00 through 99 that identifies the CA Culprit report.
- *Sss*, coded in columns 5 through 7, specifies a 3-digit number indicating the sequence number of the type 7 parameter.
- **Message-qv** (ARG1) requires the name of a 25-byte work field or an alphanumeric literal, enclosed in single quotation marks, for the message to be sent.
- **Response-v** (ARG2) requires the name of a 25-byte alphanumeric work field that receives the console operator's response.

### Example

This example shows the CA Culprit parameters required to send a message to the console operator and receive a response.

The following code:

- Defines work fields (MSGTO and MSGFROM) as the message areas
- Issues a CALL to CULLUS40 to send and receive messages
- Tests the response and specifies an action to take based upon the response

```
INPUT 80 F 4000
REC FLD1 1 50
010 MSGTO 'TYPE STOP TO DISCONTINUE'
010 MSGFROM '
.
.
.
017001 CALL US40 (MSGTO MSGFROM)
017010 IF MSGFROM EQ 'STOP' STOP
```

## Moving Fields to an Input Buffer Area (CULLUS43)

### What You Can Do

You can use CULLUS43 to move data to and from any field known to CA Culprit, including input buffer fields. CULLUS43 is particularly useful in building a table in a dummy input buffer area.

### How to Use CULLUS43

To invoke CULLUS43:

1. **Define the field to be moved** by using a REC parameter (input field) or a work field parameter.
2. **Define a dummy buffer area** to receive the data.
  - For non-database runs, use the MB= option on an additional INPUT parameter.
  - For database runs, use one INPUT parameter that includes the extra storage requirement in the record size specification.

To build a table, code REC parameters that define multiply-occurring data groups.

3. **Issue a CALL to CULLUS43** from type 7 logic:

width=80

Col

2

▼

RPT-nn7sss CALL US43 (field-name result1-v length-vn)

- *Rpt-nn*, coded in columns 2 and 3, specifies a 2-digit number in the range 00 through 99 that identifies the CA Culprit report.
- *Sss*, coded in columns 5 through 7, specifies a 3-digit number indicating the sequence number of the type 7 parameter.
- **Field-name** (ARG1) requires the name of the input or work field that is to be moved.
- **Result-v** (ARG2) requires the name of the receiving field.
- **Length-vn** (ARG3) requires a numeric literal or the name of an 8-byte packed decimal work field whose value equals the number of characters moved.

## Helpful Hint

To avoid overwriting input data when moving fields to the input buffer, the receiving field should be a dummy area whenever possible.

### Example

This example shows the CA Culprit parameters required to move data from a work field to a table in the dummy input buffer.

The following code:

- Defines the field to be moved (NAME) as an input field on a REC parameter
- Defines a dummy input buffer area to receive the data
- Defines a repeating group (TABLE) and the element (CHAR) as the receiving field in the dummy input buffer area
- Issues a CALL to CULLUS43, which uses a numeric literal (25) to specify the length of the data to move
- Prints the first and eleventh characters of the name

```
INPUT 200 F 200
REC NAME 5 25
INPUT 25 F 25 MB=D
REC TABLE 1 GROUP DD 1.25
REC CHAR 1 1 ELMNT DD
093CULLUS43
0951*005 NAME
0951*010 CHAR.1
0951*020 CHAR.11
097025 CALL US43 (NAME CHAR.1 25)
```

| REPORT NO. 09 | CULLUS43   | mm/dd/yy | PAGE | 1 |
|---------------|------------|----------|------|---|
| TERRY         | JANENS     | T        | J    |   |
| JOE           | NGUYA      | J        | N    |   |
| MARK          | TIME       | M        | T    |   |
| ROGER         | WILCO      | R        | W    |   |
| ALBERT        | BREEZE     | A        | B    |   |
| CAROLYN       | CROW       | C        | C    |   |
| BURT          | LANCHESTER | B        | L    |   |
| RENE          | MAKER      | R        | M    |   |

## Moving Variable-length Data (CULLUS45)

### What You Can Do

You can use CULLUS45 to move variable-length data, such as input record trailers, from one location to another. You can move incoming variable-length data to a series of fixed-length work fields or to a fixed part of a dummy input buffer area.

CULLUS45 has the following special features:

- It can send data to a receiving field that is larger than the sending field by using a fill character to occupy the unused portion.
- It can manipulate the sending and receiving fields without repeatedly specifying explicit data locations. CULLUS45 can keep its place across repeated fields after the location of the initial sending and receiving field is established.
- It will access previous values of an argument if four cent signs enclosed in single quotation marks ('cccc') are coded in the argument list. This is particularly useful if the argument value has been changed to accommodate some other procedure module.

**Note:** The hex value of 'cccc' must be x'4A4A4A4A'. Some non-U.S. character sets default the cent sign to an alternate hex value.

### How to Use CULLUS45

To invoke CULLUS45:

1. **Code INPUT, REC, and work field parameters.**
2. **Issue a CALL to CULLUS45** from type 7 logic:

Col  
2  
↓  
RPT-*nn*7*sss* **CALL US45** (*send-field-qv* *receive-field-qv* *send-length-qvn* [*receive-length-vn* *fill-character-q*] )

- *Rpt-nn*, coded in columns 2 and 3, specifies a 2-digit number in the range 00 through 99 that identifies the CA Culprit report.
- *Sss*, coded in columns 5 through 7, specifies a 3-digit number indicating the sequence number of the type 7 parameter.

- **Send-field-qv** (ARG1) requires the name of the field to be moved. The end of the previous location, if a call to CULLUS45 has occurred, is retrieved by using 'ccc'.
- **Receive-field-qv** (ARG2) requires the name of the receiving field. The previous value, if a value change has occurred, can be retrieved by using 'ccc'.
- **Send-length-qvn** (ARG3) requires a numeric literal or the name of an 8-byte packed decimal work field that specifies the length of the sending field. 'ccc' can be used to retrieve the previous value if a value change has occurred.
- **Receive-length-vn** (ARG4) is an optional argument that requires a numeric literal or the name of an 8-byte packed decimal work field that specifies the length of the receiving field.

If *receive-length-vn* is omitted and the internal value is 0, the value of *send-length-qvn* is used.

- **Fill-character-q** (ARG5) is an optional argument that requires a 1-byte alphanumeric character, enclosed in single quotation marks, as a filler for empty spaces in receiving fields. The default is a blank.

## Helpful Hints

- To avoid overwriting input data, use a dummy area if you move fields to the input buffer.
- If the sending field is longer than the receiving field, transfer of data stops when the receiving field is filled.
- If the receiving field length is 0, no data is moved.
- Arguments can be omitted. CULLUS45 substitutes a default value or the results of a prior CALL when an argument is missing.

### Example 1—Moving Data from a Field to a Buffer

Moving data from a work field to a dummy input buffer:

The following code:

- Defines the input file and a dummy input buffer area on INPUT parameters
- Defines the receiving field in the dummy input buffer area
- Issues a CALL to CULLUS45 to move the contents of the work field (SEND) to the input buffer dummy area

```

INPUT 200 F 200
REC NAME 5 25
INPUT 20 F 20 MB=D
REC RECEIVE 1 20
090UT 60
093CULLUS45
090 SEND 'WORK FIELD DATA'
0951*010 NAME
0951*020 RECEIVE
097030 CALL US45 (SEND RECEIVE 15 20 ' ')

```

| REPORT NO. 09 |            | CULLUS45        | mm/dd/yy | PAGE | 1 |
|---------------|------------|-----------------|----------|------|---|
| TERRY         | JANENS     | WORK FIELD DATA |          |      |   |
| JOE           | NGUYA      | WORK FIELD DATA |          |      |   |
| MARK          | TIME       | WORK FIELD DATA |          |      |   |
| ROGER         | WILCO      | WORK FIELD DATA |          |      |   |
| ALBERT        | BREEZE     | WORK FIELD DATA |          |      |   |
| CAROLYN       | CROW       | WORK FIELD DATA |          |      |   |
| BURT          | LANCHESTER | WORK FIELD DATA |          |      |   |

### Example 2—Multiple Moves with Trailers

This example moves a variable portion of an input record into a series of work fields and a dummy input buffer area.

The following code:

- Defines the following input fields:
  - The total length (TLEN) of all trailers on the record
  - Three 1-byte binary fields (L1, L2, L3) that specify the lengths (not exceeding 25-bytes) of each name and address trailer
  - Variable fields, beginning in position 100, that consist of up to 3 name and address trailers followed by 1 to 10 other trailers (not exceeding 100 bytes).

Trailers following the names and addresses are identified by one of ten possible 1-byte codes located in the first byte of the trailer.
- Defines a dummy buffer area containing:
  - 100 bytes to receive the deblocked trailers
  - The trailer identification

- Defines work fields to contain:
  - A table of trailer codes
  - A table of trailer lengths
  - An index for a table search (IX)
  - The maximum number of entries for the table (IXLIM)
  - A counter for deblocking trailers (BYTES-MVD)
  - Work areas for name and address segments
- Issues CALLs to CULLUS45 in type 7 logic to:
  - Move name and address fields into a series of work areas (NAWORK)
  - Move trailers following the name and address fields into a fixed portion of the input buffer where the data can be processed
- Checks for errors. If an error is found, an abend and a dump are forced by calling CULLUS99.

```

INPUT 400 F 800
REC TLEN 10 2 1 $Trailer length
REC L1 12 1 1 $Length of name and address trailer 1
REC L2 13 1 1 $Length of name and address trailer 2
REC L3 14 1 1 $Length of name and address trailer 3
REC TSTART 100 1 $Trailers start here
INPUT 100 MB=D
REC TRAILER 1 100 $Trailers are deblocked to this location
REC TRAILER-ID 1 1 $Reference for trailer-id
GW0 NULL '$$$' $Tells CULLUS 45 to use saved internal value
GW0 IX $Used as the index for the table search
GW0 IXLIM 10 $10 table entries
GW0 BYTES-MVD 0 $Keeps track of the number of bytes moved
GW0 WORK 0 $Sending length
GW0 NAWORK1 ' ' $5
GW0 NAWORK2 ' ' $ work areas
GW0 NAWORK3 ' ' $ for

```

```
GW0 KEY.10 '0' '1' '2' '3' '4' '5' '6' '7' '8' '9' $Trailer-ids
GW0 VAL.10 23 20 28 19 32 99 17 6 12 14 $Trailer lengths
017002 M 0 BYTES-MVD $Initialize
017030 M L1 WORK $Amount to send
017050 CALL US45 (TSTART NAWORK1 WORK 25 ' ') $Move
017070 M L2 WORK $Amount to send
017090 CALL US45 (NULL NULL WORK 25 ' ') $Move
017100 M L3 WORK $Amount to send
017130 CALL US45 (NULL NULL WORK 25 ' ') $Move
017220 COMPUTE L1 + L2 + L3 BYTES-MVD $ Set 'BYTES-MVD'
017325 CALL US45 (NULL TRAILER 1 1) $Move
017333 COMPUTE BYTES-MVD + 1 BYTES-MVD $ Adjust 'BYTES-MVD'
017345 M 1 IX $Initialize 'IX'
017350 IF KEY.IX = TRAILER-ID 375 $Branch on a hit
017355 IF IX = IXLIM 901 $Go abend
017360 COMPUTE IX + 1 IX $Increment 'IX'
017365 B 350 $Loop
017375 COMPUTE BYTES-MVD + VAL.IX BYTES-MVD $Set post-move value
017380 IF BYTES-MVD GT TLEN 901 $Error
017400 CALL US45 (NULL NULL VAL.IX 99 ' ') $Move

017412 $PROCESSING FOR THIS DEBLOCKED TRAILER GOES HERE
017415 IF BYTES-MVD LT TLEN 325 $Loop if not done
017420 IF BYTES-MVD = TLEN DROP $Drop if done
017901 CALL US99 $Abend
```

## String Search (CULLUS46)

### What You Can Do

CULLUS46 allows you to count the number of bytes contained in a string. The result is returned to a work field, which can be used for further processing.

Typically, CULLUS46 is used in conjunction with CULLUS45 to determine the length of the sending field (ARG3 of CULLUS45).

## How to Use CULLUS46

To invoke CULLUS46:

1. **Define the field containing the string to be searched** on a REC parameter or work field, as appropriate.
2. **Define a work field to receive the number of bytes counted.**
3. **Issue a CALL to CULLUS46** in type 7 logic:

Col

2

▼

RPT-nn7sss CALL US46

(**search-field-name** **search-character-qv** **range-name**  
**search-length-vn**)

- *Rpt-nn*, coded in columns 2 and 3, specifies a 2-digit number in the range 00 through 99 that identifies the CA Culprit report.
- *Sss*, coded in columns 5 through 7, specifies a 3-digit number indicating the sequence number of the type 7 parameter.
- **Search-field-name** (ARG1) requires the name of the input or work field that contains the field to be searched.
- **Search-character-qv** (ARG2) requires an alphanumeric literal, enclosed in single quotation marks, that specifies the target character for the search.
- **Range-name** (ARG3) requires the name of an 8-byte numeric work field that receives the number of characters searched. A value of -1 indicates that no search character is found.
- **Search-length-vn** (ARG4) requires a numeric literal or the name of an 8-byte numeric work field that specifies the number of bytes to search before encountering the target character (*search-character-qv*).

## Helpful Hints

CULLUS46 counts from the first character in the string up to, but not including, the delimiter. If a delimiter is not found, the search fails and a value of -1 is returned to the designated work field.

### Example 1—Using CULLUS46 Only

This example searches a 25-character input name field for the last name. The number of characters found in the last name is returned to a work field.

The following code:

- Defines the input field to be searched (NAME-LAST-FIRST) on a REC parameter
- Defines a work field (W-LEN) to receive the number of characters searched before the delimiting character is found
- Issues a CALL to CULLUS46 to search the 25-character field for a blank, which separates the last and first name in the input record and to return the number of characters searched to the work field W-LEN

```
INPUT 80 F 80
REC NAME-LAST-FIRST 1 25
520UT 60 D
520 W-LEN 0
523CULLUS46
5251*010 NAME-LAST-FIRST HH 'NAME'
5251*020 W-LEN HH 'LENGTH OF LAST NAME'
527010 CALL US46 (NAME-LAST-FIRST ' ' W-LEN 25)
527020 IF W-LEN EQ -1 100
527100 $ERROR ROUTINE
```

| REPORT NO. 52    | CULLUS46            | mm/dd/yy | PAGE | 1 |
|------------------|---------------------|----------|------|---|
| NAME             | LENGTH OF LAST NAME |          |      |   |
| JONES MARY       | 5                   |          |      |   |
| SMITH PETER      | 5                   |          |      |   |
| BROWN JACK       | 5                   |          |      |   |
| MACINTOSH JUNE   | 9                   |          |      |   |
| RICHARDS MICHAEL | 8                   |          |      |   |

**Example 2—Using CULLUS46 Output for CULLUS45 Input**

This example shows the CA Culprit parameters used to search for first and last names on input records by invoking CULLUS46 to supply the sending string length to CULLUS45 (variable-length move). CULLUS45 then moves the string to a work field.

The following code:

- Defines a 40-character input name field (NAME-LAST-FIRST)
- Defines work fields to receive:
  - The first and last name (FIRST-NAME and LAST-NAME)
  - The number of characters searched (RANGE-LTH)
  - The length of the searched area (SEARCH-LTH)
- Sets the initial value of the search area to 40 characters
- Issues CALLS to CULLUS46 to search for the number of characters in the last name and then to search for the number of characters in the first name
- Adjusts the size of RANGE-LTH to include the delimiting space
- Issues calls to CULLUS45, using CULLUS46 results, to move the last name and then the first name to work fields

CULLUS45 sets the value of the sending field (ARG1) internally so that CULLUS46 can resume the character search immediately after the last character moved.

```

INPUT 400 F 4000
REC NAME-LAST-FIRST 225 40
530 FIRST-NAME ' ' $ LENGTH = 20
530 LAST-NAME ' ' $ LENGTH = 20
530 RANGE-LTH 0 $ RANGE LENGTH
530 SEARCH-LTH 40 $ SEARCH LENGTH
.
.
.
537105 CALL US46 (NAME-LAST-FIRST ' ' RANGE-LTH SEARCH-LTH)
537110 IF RANGE-LTH NE -1 115
537 MOVE 19 TO RANGE-LTH
537115 RANGE-LTH A 1 RANGE-LTH
537120 CALL US45 (NAME-LAST-FIRST LAST-NAME RANGE-LTH 20 ' ')
537125 $ RESUME SEARCH AFTER MOVE OF LAST NAME
537 40 S RANGE-LTH SEARCH-LTH
537 CALL US46 (NAME-LAST-FIRST ' ' RANGE-LTH SEARCH-LTH)
537135 IF RANGE-LTH NE -1 140
537 MOVE 19 TO RANGE-LTH
537140 RANGE-LTH A 1 RANGE-LTH
537145 CALL US45 ('&&&&' FIRST-NAME RANGE-LTH 20 ' ')

```

## Creating a Run-time Message (CULLUS48)

### What You Can Do

You can use CULLUS48 to add your own message to the Run-Time Message Section of CA Culprit output. The addition of status or diagnostic messages can be used to track processing flow and facilitate debugging.

### How to Use CULLUS48

To invoke CULLUS48:

1. **Code INPUT and REC parameters**, as needed.
2. **Issue a CALL to CULLUS48:**

Col

2

▼

RPT-**nn**7**sss** CALL US48 ( **message-qv**)

- *Rpt-nn*, coded in columns 2 and 3, specifies a 2-digit number in the range 00 through 99 that identifies the CA Culprit report.
- *Sss*, coded in columns 5 through 7, specifies a 3-digit number indicating the sequence number of the type 7 parameter.
- **Message-qv** (ARG1) requires the name of an alphanumeric work field that contains the message enclosed in single quotation marks. The message must:
  - Begin with a carriage control character.
  - End with two percent signs (%%). Use %% exclusively to terminate messages.
  - Not exceed 132 characters.

#### Example

The following example shows the CA Culprit parameters required to add a message to the Run-Time Message Section of a CA Culprit job.

The following code:

- Tests for the value of the field (SAMPLE) that triggers the message
- Issues a CALL to US48 if the test fails:
  - 0 is the carriage control character that precedes the message.
  - %% signals the completion of the message.

```

IN 80 F 80
REC SAMPLE 1 2 2
520UT 60 D
523CULLUS48
5251*010 SAMPLE HH 'SAMPLE COUNT'
527010 IF SAMPLE NE 0 020
527 CALL US48 ('0US48 -- NO SAMPLE: RUN STOPS%%')
527015 STOP
527020 TAKE

```

|                              |                   |        |      |   |
|------------------------------|-------------------|--------|------|---|
| mm/dd/yy                     | RUN TIME MESSAGES | volser | PAGE | 1 |
| US48 -- NO SAMPLE: RUN STOPS |                   |        |      |   |
| ***** END OF FILE *****      |                   |        |      |   |
| 3 INPUT RECORDS READ         |                   |        |      |   |

## Converting Binary Strings (CULLUS50)

### What You Can Do

You can use CULLUS50 to convert a binary string to an alphanumeric string or a series of 8-byte packed decimal numeric values. This is especially useful when bit flags need to be converted into a format suitable for testing in CA Culprit code.

### How to Use CULLUS50

To invoke CULLUS50:

1. **Define the binary input field** on a REC parameter.
2. **Define a work field** to receive the result of the conversion.
  - A non-subscripted work field must be equal in length to the number of bits targeted for conversion.
  - A subscripted work field must occur the same number of times as the number of bits targeted for conversion.
3. **Define a work field to hold the number of bits targeted for conversion** unless you use a numeric literal.
4. **Issue a CALL to CULLUS50:**

Col

2

▼

RPT-nn7sss CALL US50 (start-bit-field target-field-name bit-count-vn)

- *Rpt-nn*, coded in columns 2 and 3, specifies a 2-digit number in the range 00 through 99 that identifies the CA Culprit report.
- *Sss*, coded in columns 5 through 7, specifies a 3-digit number indicating the sequence number of the type 7 parameter.
- **Start-bit-field** (ARG1) requires the name of the input or work field to be converted.
- **Target-field-name** (ARG2) requires the name of the field receiving the conversion.
- **Bit-count-vn** (ARG3) can be a numeric literal or the name of an 8-byte numeric work field that specifies the number of bits targeted for conversion.

### Example

This example shows the CA Culprit parameters required to convert a binary input field to alphanumeric and packed decimal format.

The following code:

- Defines the binary field (RDW) on a REC parameter
- Defines a 16-byte alphanumeric work field to receive the binary number
- Defines a subscripted work field to receive the binary values and store them in a numeric table
- Issues a CALL to CULLUS50 to perform each conversion. A numeric literal (16) specifies the number of bits converted

```

INPUT 280 V 280
REC RDW 1 4 1
010 ALPHA '1234567890123456'
010 NUMER.16 2
013CULLUS50
0151*001 ALPHA HH 'ALPHA'
0151*010 NUMER.1 FM '9' HH 'DIGIT 1'
0151*020 NUMER.12 FM '9' HH 'DIGIT 12'
0151*030 NUMER.13 FM '9' HH 'DIGIT 13'
0151*040 NUMER.14 FM '9' HH 'DIGIT 14'
017010 CALL US50 (RDW ALPHA 16)
017020 CALL US50 (RDW NUMER.1 16)

```

| REPORT NO. 01    | CULLUS50 |          | mm/dd/yy |          | PAGE | 1 |
|------------------|----------|----------|----------|----------|------|---|
| ALPHA            | DIGIT 1  | DIGIT 12 | DIGIT 13 | DIGIT 14 |      |   |
| 0000000100010100 | 0        | 1        | 0        | 1        |      |   |

## Concatenating Fields (CULLUS53)

### What You Can Do

You can use CULLUS53 to concatenate up to three input fields and store them in one work field for processing. For example, you can concatenate name and address fields for printing labels.

### How to Use CULLUS53

To invoke CULLUS53:

1. **Define the Input Fields to be concatenated.**
2. **Define numeric work fields** that contain the length of the sending and receiving fields if numeric literals are not used.
3. **Define an alphanumeric work field** to receive the concatenation.
4. **Issue a CALL to CULLUS53** from type 7 logic:

Col

2

▼

```
RPT-nn7sss CALL US53 (first-field-v first-field-length-vn second-field-v
second-field-length-vn third-field-v third-field-vn
comb-field-v comb-field-length-vn)
```

- *Rpt-nn*, coded in columns 2 and 3, specifies a 2-digit number in the range 00 through 99 that identifies the CA Culprit report.
- *Sss*, coded in columns 5 through 7, specifies a 3-digit number indicating the sequence number of the type 7 parameter.
- **First-field-v** (ARG1) requires the name of the first alphanumeric field for concatenation.
- **First-field-len-vn** (ARG2) requires a numeric literal or the name of a numeric work field that specifies the length of the first field to be concatenated (*first-field-v*).
- **Second-field-v** (ARG3) requires the name of the second alphanumeric field for concatenation.
- **Second-field-len-vn** (ARG4) requires a numeric literal or the name of a numeric work field that specifies the length of the second field to be concatenated (*second-field-v*).
- **Third-field-v** (ARG5) requires the name of the third alphanumeric field for concatenation.
- **Third-field-len-vn** (ARG6) requires a numeric literal or the name of a numeric work field that specifies the length of the third field to be concatenated (*third-field-v*).
- **Comb-field-v** (ARG7) requires the name of the alphanumeric field receiving the concatenation.
- **Comb-field-len-vn** (ARG8) requires a numeric literal or a numeric work field that specifies the length of the field receiving the concatenated value (*comb-field-v*).

## Helpful Hints

- CULLUS53 automatically eliminates trailing blanks attached to individual data segments.
- Blank fields and fields with zero length are ignored.
- Data returned to the receiving field is compressed. Data segments are separated by a blank.
- The receiving field is padded with blanks to the right of the last significant character. If you don't want the field padded with blanks, make the length of the receiving field equal to the sum of the length of the data segments plus one space between each field, but not more than 132.

### Example

This example concatenates name and address fields to produce a mailing list.

The following code:

- Defines input data fields
- Defines work fields to receive the name (FULL-NAME) and the address (CITY-STATE-ZIP)
- Issues a CALL to CULLUS53 to concatenate the elements of the name and another CALL to concatenate the elements of the address

```

INPUT 80 F 400
REC NAME 6 26
REC FIRST-NAME 6 10
REC LAST-NAME 16 10
REC STREET 26 20
REC CITY 46 15
REC STATE 61 2
REC ZIP 63 5
093CULLUS53
090 FULL-NAME '
090 CITY-STATE-ZIP '
090 BLANK ' '
09510001 FULL-NAME
09520001 STREET
09530001 CITY-STATE-ZIP
09540001 BLANK
097010 CALL US53 (FIRST-NAME,10, LAST-NAME,10,BLANK,1,
*FULL-NAME,23)
097025 CALL US53 (CITY 15 STATE 2 ZIP 5
* CITY-STATE-ZIP 24)

```

|                                                                                                                           |          |          |        |
|---------------------------------------------------------------------------------------------------------------------------|----------|----------|--------|
| REPORT NO. 09                                                                                                             | CULLUS53 | mm/dd/yy | PAGE 1 |
| AMOS JOHNSON<br>22651 MASS AVENUE<br>SAN FRANCISCO CA 09801<br><br>BRUCE THORPE<br>11002 PEACHTREE LA<br>ATLANTA GA 76543 |          |          |        |

## Searching a Table (CULLUS62)

### What You Can Do

You can use CULLUS62 to search an alphanumeric or numeric table for specific values without coding a series of tests, moves, and computations in type 7 logic.

### How to Use CULLUS62

To invoke CULLUS62:

First—**Define input fields** on REC parameters.

Second—**Define required fields that are not input fields or specified by literals** on work field parameters. For example:

- The table, which is a multiply-occurring field
- The table type code
- The number of bytes in each entry of the table
- The total number of the table entries
- The type of search code
- The numeric work field that will receive the occurrence number of the key value in the table

Third—**Issue a CALL to CULLUS62** in type 7 logic:

Col

2

RPT-*nn*7*sss* CALL US62 (**table-field-name** **table-type-qv** **entry-length-vn**  
**entry-count-vn** **search-type-qv** **key-vn** **index-v**)

- *Rpt-nn*, coded in columns 2 and 3, specifies a 2-digit number in the range 00 through 99 that identifies the CA Culprit report.
- *Sss*, coded in columns 5 through 7, specifies a 3-digit number indicating the sequence number of the type 7 parameter.
- **Table-field-name** (ARG1) requires the name of the table to be searched.
- **Table-type-qv** (ARG2) requires the name of a work field or a 1-character alphanumeric code, enclosed in single quotation marks, that specifies the type of table searched. Valid codes are shown below:

| Table type... | Code... | Table entry length... |
|---------------|---------|-----------------------|
| Alphanumeric  | 'A'     | 1 through 64          |
| Numeric       | 'N'     | 8 or 16               |

- **Entry-length-vn** (ARG3) requires a numeric literal or the name of an 8-byte numeric work field that indicates the length of each table entry (see above).
- **Entry-count-vn** (ARG4) requires a numeric literal or the name of an 8-byte numeric work field that indicates the total number of entries in the table. The maximum is 32,767.
- **Search-type-qv** (ARG5) requires the name of a work field or a 1-character alphanumeric literal, enclosed in single quotation marks, that specifies the type of search:

| Search type... | Code... | Table entry order... |
|----------------|---------|----------------------|
| Binary         | 'B'     | Ascending            |
| Sequential     | 'S'     | Any order            |

- **Key-vn** (ARG6) requires a numeric literal, alphanumeric literal, or the name of a field that contains the search value.
- **Index-v** (ARG7) requires the name of a numeric work field that receives the number of the entry in the table that matches the key value (**key-vn**).

## Helpful Hints

- CULLUS62 does not convert data or align decimals. If either procedure is desired, move the field that has the key value to an appropriate work field.
- If the key value is invalid or missing from the table, CULLUS62 automatically returns a value of zero.
- Use a key value that corresponds in data type and number of decimal places to the table entries.
- Use a sequential search when entries are not randomly distributed and where entries are not widely diversified.

### Example

This example searches a table containing six occurrences of department numbers. Department names that correspond to the department number are retrieved and printed.

The following code:

- Defines the input data fields, including the field that contains key value (IN-DEPT).
- Defines work fields containing:
  - 6 values for department number (DEPT-NUMBER)
  - 7 department names (DEPARTMENT-NAME)
  - The occurrence number of the key in the DEPT-NUMBER table (INDEX)
- Issues a CALL to CULLUS62 to search the DEPT-NUMBER table and place the occurrence number (INDEX) when a match occurs in INDEX.

The value of INDEX is tested for a return of zero (no key found). If true, a value corresponding to the seventh occurrence of DEPARTMENT-NAME is moved to INDEX and UNKNOWN is printed for department numbers not found in the DEPT-NUMBER table.

- Prints out specific occurrences of DEPARTMENT-NAME with type 5 lines.

```

INPUT 80 F 80
REC NAME 1 20
REC ACCOUNT 33 4 3
REC IN-DEPT 37 3
013CULLUS62
0141*001 ' '
0151*001 NAME HH 'NAME'
0151*002 ACCOUNT FN HH 'ACCOUNT'
0151*003 DEPARTMENT-NAME.INDEX HH 'DEPARTMENT' 'NAME'
010 DEPT-NUMBER.6 '111' '222' '333' '444' '555' '666'
010 DEPARTMENT-NAME.7 'TECH SUPPORT' 'SALES'
* 'MARKETING' 'SYSTEM SUPPORT'
* 'ADMINISTRATION' 'EDUCATION'
* 'UNKNOWN'
010 INDEX
017001 CALL US62 (DEPT-NUMBER 'A' 3 6 'S' IN-DEPT INDEX)
017 IF INDEX EQ 0 50
017 TAKE
017050 MOVE 7 TO INDEX
017 TAKE
010UT D

```

|                                          |          |                |    |
|------------------------------------------|----------|----------------|----|
| REPORT NO. 01                            | CULLUS62 | mm/dd/yy PAGE  | 1  |
|                                          |          | DEPARTMENT     |    |
| NAME                                     | ACCOUNT  | NAME           |    |
| JONES MARY                               | 1112222  | TECH SUPPORT   |    |
| SMITH PETER                              | 3334444  | MARKETING      |    |
| BROWN JACK                               | 5556666  | ADMINISTRATION |    |
| MACINTOSH JUNE                           | 7778888  | UNKNOWN        |    |
| RICHARDS MICHAEL                         | 9991111  | UNKNOWN        |    |
| PAPPAS DICK                              | 2223333  | SALES          |    |
| BURNS FAY                                | 4445555  | SYSTEM SUPPORT |    |
| C750009 RECORDS WRITTEN FOR REPORT 01 -- |          |                | 12 |

## Processing Data Dictionary Reporter Tables (CULLUS64)

### What You Can Do

You can use CULLUS64 to create and read a table of user-defined attributes and user-defined nested comments from the integrated Data Dictionary. CULLUS64 can be used if you are writing your own data dictionary reports or are modifying CA-supplied reports. See the *CA IDMS Reports Guide* for more information about CA-supplied reports.

## How to Use CULLUS64

To invoke CULLUS64:

First—**Access the database and identify the subschema** by using the DB and SS= options of the INPUT parameter.

Second—**Identify the route through the dictionary** by using the PATH parameter.

Third—**Define a numeric work field to hold the key to the user-defined attribute.**

Fourth—**Define a subscripted alphanumeric work field to hold the user-defined attributes.**

Fifth—If you do not use literals, **code alphanumeric work fields** for:

- The action code
- The entity type
- The attribute type

Sixth—**Issue a CALL to CULLUS64** from type 7 logic:

Col

2

▼

RPTnn7sss CALL US64 (action-code-qv entity-type-qv comment-next-key-qv  
comm-next-id-v table-value-v)

- *Rpt-nn*, coded in columns 2 and 3, specifies a 2-digit number in the range 00 through 99 that identifies the CA Culprit report.
- *Sss*, coded in columns 5 through 7, specifies a 3-digit number indicating the sequence number of the type 7 parameter.
- **Action-code-qv** (ARG1) requires an alphanumeric work field or 1-character alphanumeric literal, enclosed in single quotation marks, to specify the action to take:

| Code... | Action taken...  |
|---------|------------------|
| '0'     | Create the table |
| '1'     | Read the table   |

- **Entity-type-qv** (ARG2) requires an alphanumeric work field or 20-character alphanumeric literal, enclosed in single quotation marks, to indicate the class associated with the attribute and provide access to the entry. The valid classes are:

|             |                    |                     |
|-------------|--------------------|---------------------|
| SYSTEM      | PROGRAM            | MODULE              |
| FILE        | RECORD             | ELEMENT             |
| USER        | ATTRIBUTE          | CLASS               |
| MESSAGE     | TASK               | QUEUE               |
| DESTINATION | LOGICAL - TERMINAL | PHYSICAL - TERMINAL |
| LINE        | PANEL              | MAP                 |

- **Comment-nest-key-qv** (ARG3) requires an alphanumeric work field or 1-character alphanumeric literal, enclosed in single quotation marks, to indicate the type of attribute. Valid attribute codes are listed below under ARG4.
- **Comm-nest-id-v** (ARG4) requires an 8-byte numeric work field to provide a key to the user-defined attribute. Valid key formats are listed below:

| Attribute type... | Attribute code... | Key format... |
|-------------------|-------------------|---------------|
| Comment           | 'C'               | CMT-ID-nnn    |
| Nest              | 'N'               | NEST-ID-nnn   |

- **Table-value-v** (ARG5) requires a 40-character alphanumeric work field to send table updates or receive table data.

## Helpful Hints

- CA Culprit dynamically creates a single external table that can be used by one or more reports. The total number of table entries need not be known at run time.
- Be sure to move CMT-ID-**nnn** or NEST-ID-**nnn** to a numeric work field and specify that work field on the CALL statement.
- *Nnn* must be positive integers.
- CULLUS64 does not retrieve system-defined relationships.

### Example

This example shows the parameters required to produce a Data Dictionary Report based on CLASS attributes identified by the key CMT-ID-086.

The following code:

- Specifies a CA IDMS/DB database and subschemas IDMSNWKA of schema IDMSNTWK on the INPUT parameter
- Identifies the route through the data dictionary on the PATH parameter
- Issues a CALL to CULLUS64 to retrieve user-defined comment entities

```
INPUT 10000 F 10000 DB(D) SS=IDMSNWKA,IDMSNTWK
PATHA2 00AK-012 CLASS-092 CLASSCMT-086
GW0 ATTR-NAME.18 'SYSTEM '
* 'PROGRAM '
* 'MODULE '
* 'FILE '
* 'RECORD '
* 'ELEMENT '
* 'USER '
* 'ATTRIBUTE '
* 'CLASS '
* 'MESSAGE '
* 'TASK '
* 'QUEUE '
* 'DESTINATION '
* 'LOGICAL - TERMINAL '
* 'PHYSICAL - TERMINAL '
* 'LINE '
* 'PANEL '
* 'MAP '
010 CMT-ID
010 PRINT-40 '
.
.
.
017250 MOVE CMT-ID-086 TO CMT-ID
017300 $ CALL MODULE TO RETRIEVE USER-DEFINED COMMENT TYPE
017 CALL US64 ('1' ATTR-NAME.9 'C' CMT-ID PRINT-40)
```

## Memory Dump (CULLUS99)

### What You Can Do

You can use CULLUS99 to produce a region or partition dump.

## How to Use CULLUS99

To invoke CULLUS99:

- **Issue a CALL to CULLUS99** from type 7 logic:

Col

2

▼

RPT-**nn7sss** *CALLUS99*

- *Rpt-nn*, coded in columns 2 and 3, specifies a 2-digit number in the range 00 through 99 that identifies the CA Culprit report.
- *Sss*, coded in columns 5 through 7, specifies a 3-digit number indicating the sequence number of the type 7 parameter.

- **Code JCL statements** to produce a dump:

| System... | Result...                                                                                                                                                        | CULP3 JCL statement...                             |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------|
| z/OS      | An OC1 abend code                                                                                                                                                | //SYSUDUMP DD SYSOUT=A                             |
| z/VSE     | A USER 999 abend code                                                                                                                                            | // OPTION DUMP                                     |
| z/VM      | DMSABN155T<br>USER ABEND nnnn called from<br>addr<br>nnnn = abend type<br>(in this case, an<br>operation exception)<br>addr = virtual<br>address of the<br>abend | Enter the DEBUG mode and<br>issue the DUMP command |



# Chapter 4: Output Modules

---

This section contains the following topics:

[What Is an Output Module?](#) (see page 119)

[What You Can Do with an Output Module](#) (see page 119)

[How to Invoke an Output Module](#) (see page 120)

[Formatting a Hexadecimal Buffer Dump \(CULEDUMP\)](#) (see page 120)

[Printing Labels \(CULELABEL\)](#) (see page 123)

[Printing Multiple Lines \(CULEMLIN\)](#) (see page 126)

[Writing Formatted Records to a VSAM File \(CULEVSAM\)](#) (see page 136)

[Segmenting Reports in a VSE/POWER Run \(CULEPOWR\)](#) (see page 138)

## What Is an Output Module?

An output module is a subroutine called during the CULE processing phase of a CA Culprit job to facilitate special output formatting.

## What You Can Do with an Output Module

The tasks you can perform with CA-supplied output modules are listed in the following table.

### What You Can Do with CA-supplied Output Modules

| To...                                                                                         | Use...    |
|-----------------------------------------------------------------------------------------------|-----------|
| Format a vertical or horizontal hexadecimal buffer dump for fixed- or variable-length records | CULEDUMP  |
| Format sorted or unsorted 1- to 8-line labels                                                 | CULELABEL |
| Print more than eight heading, detail, and footer lines                                       | CULEMLIN  |
| Write formatted records to an existing VSAM file                                              | CULEVSAM  |
| Produce printed or punched output for specific CA Culprit reports through VSE/POWER           | CULEPOWR  |

## How to Invoke an Output Module

Output modules are invoked by the **UM** option of the **OUTPUT** parameter:

**Col**  
**2**  
↓  
**RPT-*nn* OUTPUT UM (*module-name* [ , *special-value-a* ] )**

- *Rpt-nn*, coded in columns 2 and 3, specifies a 2-digit number in the range 00 through 99 that identifies the CA Culprit report.
- *Module-name* requires an 8-character name of the output module invoked.
- *Special-value-a* requires a 1- or 2-character code specific to some (not all) output modules. If present, this value is preceded by a comma.

## Formatting a Hexadecimal Buffer Dump (CULEDUMP)

CULEDUMP produces a hexadecimal dump of the CA Culprit output buffer in horizontal or vertical format, as shown below.

### Horizontal and Vertical Dump Formats

In horizontal (default) format, hexadecimal representation of data precedes EBCDIC characters for the same data; both sets of data print on the same line. The first two columns of the dump define the position of the first character of data shown on that line in the output buffer and the address of that line in storage. In vertical format, EBCDIC characters print immediately above their hexadecimal representation. The bottom line of this dump indicates the position in the output buffer, but storage address information is not available.

### Horizontal Dump:

| 1 INPUT RECORDS READ                       |         |          |          |          |          |          |          |          |          |   |                          |   |   |   |
|--------------------------------------------|---------|----------|----------|----------|----------|----------|----------|----------|----------|---|--------------------------|---|---|---|
| HEX DUMP OUTPUT                            |         |          |          |          |          |          |          |          |          |   |                          |   |   |   |
| POSITION                                   | ADDRESS | STORAGE  |          |          |          |          |          |          |          |   |                          |   |   |   |
| 00001                                      | 00AB28  | 40003200 | 00E3C8C9 | E240C9E2 | 40F3F240 | C2E8E3C5 | E240D3D6 | D5C74040 | 40404040 | * | ...THIS IS 32 BYTES LONG | * |   |   |
| 00033                                      | 000020  | 40404040 | 40404040 | 40404040 | 40404040 | 40404040 | 40404040 | 40404040 | 40404040 | * |                          | * |   |   |
| 00065                                      | 000040  | 40404040 | 40404040 | 404040F1 | F16BF0F0 | F0404040 | 40404040 | 40404040 | 40404040 | * | 11,000                   | * |   |   |
| 00097                                      | 000060  | 40404040 | 40       |          |          |          |          |          |          |   |                          |   | * | * |
| C750009 RECORDS WRITTEN FOR REPORT 01 -- 1 |         |          |          |          |          |          |          |          |          |   |                          |   |   |   |



- *Variable-length-indicator* requires a V as a 1-character code to specify a variable-length record dump. The default is a dump of the entire output buffer for each record.

## Helpful Hints

- The dump format option for CULEDUMP is independent of the PROFILE parameter HD= option. The HD= option applies only to dumps produced by the extended error-handling facility. See *CA Culprit for CA IDMS Messages and Codes Guide* for more information.
- When the variable-length indicator is used, the length of the record (RDW) must be contained in the first 4 bytes of the output buffer.

### Example 1—Horizontal Dump of Fixed-length Records

This example produces a horizontal dump of fixed-length output buffer records.

The following code:

- Defines an 80-byte input record as one field (FLD)
- Uses the OUTPUT parameter to:
  - Allocate 100 bytes for each output record
  - Request a details only report
  - Issue a CALL to CULEDUMP to create a horizontal (default) dump
- Specifies the output (FLD) on a type 5 line using exact column placement.

```
IN 80 F 80
REC FLD 1 80
01OUT 100 D UM(CULEDUMP)
01510001 FLD
```

| 1 INPUT RECORDS READ                       |         |          |          |          |          |          |          |          |          |                            |
|--------------------------------------------|---------|----------|----------|----------|----------|----------|----------|----------|----------|----------------------------|
| HEX DUMP OUTPUT                            |         |          |          |          |          |          |          |          |          |                            |
| POSITION                                   | ADDRESS | STORAGE  |          |          |          |          |          |          |          |                            |
| 00001                                      | 00AB28  | 40003200 | 00E3C8C9 | E240C9E2 | 40F3F240 | C2E8E3C5 | E240D3D6 | D5C74040 | 40404040 | * ...THIS IS 32 BYTES LONG |
| 00033                                      | 000020  | 40404040 | 40404040 | 40404040 | 40404040 | 40404040 | 40404040 | 40404040 | 40404040 | *                          |
| 00065                                      | 000040  | 40404040 | 40404040 | 404040F1 | F16BF0F0 | F0404040 | 40404040 | 40404040 | 40404040 | * 11,000                   |
| 00097                                      | 000060  | 40404040 | 40       |          |          |          |          |          |          | *                          |
| C750009 RECORDS WRITTEN FOR REPORT 01 -- 1 |         |          |          |          |          |          |          |          |          |                            |

Example 2—Vertical Dump of Variable-length Records

This example produces a vertical dump of variable-length output buffer records.

The following code:

- Defines the largest record length in the file as 80 bytes
- Defines the file as one field (FLD)
- Uses the OUTPUT parameter to:
  - Allocate a maximum of 100 bytes for the output buffer
  - Specify a details-only report
  - Issue a CALL to CULEDUMP to create a vertical dump (V)
- Uses a type 5 parameter with exact column placement to place the FLD in the output buffer

```
IN 80 F 80
REC FLD 1 80
01OUT 100 D UM(CULEDUMP VV)
01510001 FLD
```

|                                            |                                                        |
|--------------------------------------------|--------------------------------------------------------|
| 1 INPUT RECORDS READ                       |                                                        |
| HEX DUMP OUTPUT                            |                                                        |
| CHAR                                       | THIS IS 32 BYTES LONG                                  |
| ZONE                                       | 0300ECCCE4CE4FF4CEECE4DDDC4444444444444444444444444444 |
| DIGIT                                      | 02003892092032028352036570000000000000000000000000000  |
|                                            | 01...5...10...5...20...5...30...5...40...5...50        |
| C750009 RECORDS WRITTEN FOR REPORT 01 -- 1 |                                                        |

Printing Labels (CULELABEL)

What You Can Do

You can use CULELABEL to print 1- to 8-line labels, in sorted or unsorted sequence, on regular or special forms.

## How to Use CULELABL

Supply a carriage control tape or a function control block (FCB) to direct channel 1 to the first line of each label.

To invoke CULELABL:

1. Use a SORT parameter if you want to print the labels in alphabetical order.
2. Code type 5 or type 6 edit parameters with:
  - **The label information.**
  - **A carriage control character of 1** in column 10 for the first edit line.
  - **The exact start position of each detail line**, based on the record size defined on the output parameter. See the following diagram for calculating the record and block size.
3. **Specify CULELABL on the OUTPUT parameter**, using the UM option:

Col

2

▼

RPT-*nn*OUTPUT **record-size-n block-size-n**UM(CULELABL)

- *Rpt-nn*, coded in columns 2 and 3, is a 2-digit report number in the range 00 through 99.
- **Record-size-n** requires the number of print positions occupied by each label on a page. Include blank spaces before and after printed information (see the following diagram).
- **Block-size-n** requires the number of print positions occupied by all labels across the page. To determine **block-size-n**, multiply **record-size-n** by the number of labels (see the following diagram). This specification should not exceed the printer's output line size (usually 132 characters).

### CULELABL Record Size and Block Size Calculation

The following diagram shows the CULELABL record size and block size calculation.



**BLOCKSIZE = RECORD SIZE X N (NUMBER OF LABELS)**

### Example

This example reads 80-character input records and prints name and address labels on a form having two labels across the page and several labels down the page.

The following code:

- Uses the OUT parameter to specify:
  - A record size of 34 characters
  - A block size of 68 characters to accommodate two labels across the page
  - A CALL to CULELABL
- Uses a carriage control character of 1 in column 10 for the first line of each new label

```
INPUT 80 F 80
99OUT 34 68 D UM(CULELABL)
REC NAME 1 25
REC STREET-NAME 26 19
REC STATE-CODE 45 2
REC CITY-NAME 47 10
REC ZIP-CODE 57 9
995100011NAME
99520001 STREET-NAME
99530001 CITY-NAME
99530012 STATE-CODE
99530015 ZIP-CODE
```

#### An Example of Name and Address Labels

**GERALDINE A. BUTZ**  
**429 EAST 301 ST**  
**RANDOLPH KS 66554**

**DAVID E. GUY**  
**7737 BOSTON RD.**  
**FITZWILLIAM NH 03447**

**WILLIAM WELLIN**  
**91 EDWARDS ROAD**  
**FREEPORT MI 49325**

**BRUCE H. YELLIN**  
**990 OLDTOWNE RD**  
**PEORIA IL 61603**

**SHEILA FISHER**  
**173 AVON CIRCLE**  
**CHICAGO IL 60601**

**KENNETH J. CRANDALL**  
**44 DARTMOUTH ST**  
**RHINEBECK NY 12572**

**DIANE FISHER**  
**99 PRINCETON ST**  
**CEDARHURST NY 11516**

**K. J. CRANDELL PRIN.**  
**258 YALE STREET**  
**BINGHAMTON NY 13901**

**FRANCISCO BUNNELL**  
**133 ASYLUM AVE**  
**HEMLOCK IN 46937**

**FRANCIS J. TAYLOR**  
**44 RIDGE ROAD**  
**LOS ANGELES CA 90026**

**KATHLEEN F. SCHWARTZ**  
**P.O. BOX 4017**  
**BAYONNE NJ 07002**

**GENE GAMBON**  
**R.F.D. 3 BOX 13**  
**MONTGOMERY AL 36111**

## Printing Multiple Lines (CULEMLIN)

### What You Can Do

You can use CULEMLIN to print out more than eight header, detail, total, or footer lines. Reports can contain heading and footer lines on any page and correspondence, such as confirmation letters, can include data read in from the input file.

## How It Works

CULEMLIN extends CA Culprit's ability to print more than eight output lines by dividing the output buffer contents into equal segments that correspond to the length of the print line. More than one segment, not exceeding the specified length of the output buffer, can be coded on the same detail line of a type 4, 5, or 6 edit parameter.

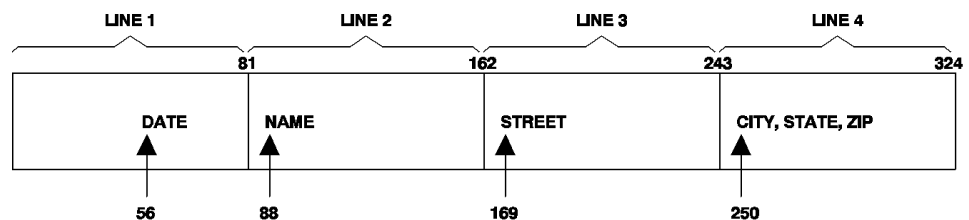
The process followed by CA Culprit and CULEMLIN is:

- **CA Culprit:**
  - Determines the size of the output buffer from the specification given on the OUTPUT parameter
  - Constructs the contents of the output buffer at output time from data entered on type 4, 5, and 6 edit parameters
  - Passes the contents of the output buffer to CULEMLIN
- **CULEMLIN:**
  - Divides the contents of the output buffer into equal segments. Each segment equals 81 or 133 characters, depending on printer requirements.
  - Prints each segment.

## How CULEMLIN Works

The output buffer has a defined length of 324 characters, which is evenly divided into segments of 81 bytes (80 text plus 1 carriage control character). The code specifies the exact placement of the text of each line.

The following figure shows the output buffer format (CULEMLIN):



Shown below is the code specifying text placement:

```
10510001 '1'
10510056 'SEPTEMBER 27, 1999'
10510082 ' - '
10510088 NAME
10510169 STREET
10510250 CITY
10510264 STATE
10510267 ZIP-CODE
```

Shown below is the printed output:

SEPTEMBER 27, 1999

AMOS        JOHNSON  
22651 MASS AVENUE  
SAN FRANCISCO CA 09801

## How to Use CULEMLIN

To invoke CULEMLIN:

First—**Define the input file.**

Second—**Specify CULEMLIN on the OUTPUT parameter** using the UM option:

**Col**  
**2**  
↓  
**RPT-*nn* OUTPUT *record-size-n* UM(CULEMLIN [ , *special-value-*nn** ] )**

- *Rpt-nn*, coded in columns 2 and 3, requires a 2-digit report number in the range 00 through 99.
- *Record-size-n* requires the number of characters held in the output buffer. This number must be a multiple of the length of the printed line plus a carriage control character (81 or 133).
- *Special-value-nn* can be:
  - An optional 2-digit number in the range 00 to 99 that sets the lines-per-page count and directs output to a file identified by the same number. The defaults are 55 lines to a page and SYS004 as the output file.
  - A 1-byte binary number followed by a space with a value between X'01' and X'3F', inclusive. This value can be submitted only from a terminal with hexadecimal input facilities. When applied as a binary number, output is directed to SYS004.

Third—**Define one page heading**, using the special character #, on a type 4 parameter.

Fourth—**Define detail and total lines** (type 5 and type 6 parameters) as required by the report. Use absolute, rather than relative, column positions.

Fifth—**Use a carriage control character** in a column that is an even multiple of the printer line size (81 or 133 characters) and less than the number of characters held in the output buffer (*record-size-n*). Valid carriage control characters are listed in the following table.

## ASA Control Characters

| Character | Description                                               |
|-----------|-----------------------------------------------------------|
| (SPACE)   | One line is advanced before printing.                     |
| 0         | Two lines are advanced before printing.                   |
| -         | Three lines are advanced before printing.                 |
| +         | No lines are advanced before printing (causes overprint). |
| 1         | Printer ejects to a new page before printing.             |
| 2-9       | Printer skips to channel 2-9 before printing.             |
| A-F       | Printer skips to channel 10-15 before printing.           |
| '#'       | Specifies heading lines on type 4 parameters.             |
| '@'       | Specifies footer lines on type 5 parameters.              |
| '*'       | Suppresses blank lines.                                   |

CULEMLIN also provides these additional options:

- **Automatic page numbering**, coded by using %PAGE, enclosed in single quotation marks, on a heading or footer line.
- **Page control breaks**, coded by using the:
  - SORT parameter
  - The carriage control character 1, enclosed in single quotation marks, on a type 4 heading line

## Helpful Hints

- A report layout form to calculate the position of the report fields is useful.
- The number of characters, including spaces, defined for each edit line must not exceed the size of the output buffer. The limit is 1330 bytes.
- Type 3 (title) parameters and autoheaders should be avoided.
- CULEMLIN holds heading and footer lines until:
  - The line-per-page count exceeds CULEMLIN's maximum (default is 55)
  - The ASA control character '1' on a detail line signals a page eject
- The default output file (SYS004) can be overridden by using the DD= clause on the OUTPUT parameter or the *special-value-nn* UM(CULEMLIN) option described above to direct the output.

### Example 1—Printing a Letter Containing Variables

This example produces confirmation letters for customers having installment loans. The account number, outstanding balance, late charges, and remaining payments vary in each letter.

The following code:

- Copies the input file definition into the code (=COPY)
- Specifies 324 bytes for the output buffer, based on the requirements of the heading (4 printed lines of 81 characters each)
- Defines the heading by using '#' on a type 4 parameter

- Creates multiple detail lines by using:
  - Five logical divisions (new buffer contents), which are identified by control characters '1' or '-' placed in the first position of the buffer line
  - Two additional divisions created by using a third and fourth type 5 detail line
- Uses control characters, not placed in the first position of the buffer line, to create spacing
- Uses blanks in the output buffer to print two spaces between the third and fourth detail lines

IN 80

```

REC ACCOUNT 1 5
REC NAME 6 18
REC STREET 24 19
REC CITY 43 13
REC STATE 56 2
REC ZIP-CODE 58 5
REC BALANCE 63 7 2 DP=2
REC ODCHG 70 5 2 DP=2
REC REMAIN 75 3 2

```

100OUT 324 D UM(CULEMLIN) \$Output buffer = 81 bytes x 4 print lines

10410001 '#' \$Identifies the heading

10410033 'LAST NATIONAL BANK'

10410115 '1234 MAIN STREET'

10410197 'SOMEWHERE, USA'

10410244 '0'

10510001 '1'

**\$New buffer contents 1**

10510056 'month dd, yyyy'

\$Print line 1

10510082 '-'

\$Triple space

10510088 NAME

\$Print line 2

10510169 STREET

\$Print line 3

10510250 CITY

\$Print

10510264 STATE

\$ line

10510267 ZIP-CODE

\$ 4

10520001 '-'

**\$New buffer contents 2**

10520007 'DEAR CUSTOMER:'

10520082 '-'

```

10520092 'FROM TIME TO TIME, AS PART OF OUR REGULAR AUDIT'
10520140 'PROCEDURE, WE'
10520169 'ASK OUR CUSTOMERS TO CONFIRM THAT'
10520203 'THEIR RECORDS ARE IN AGREEMENT'
10520250 'WITH OURS. THE INFORMATION SHOWN BELOW IS TAKEN'
10520299 'FROM OUR RECORDS'
10530007 'OF YOUR ***INSTALLMENT LOAN*** ACCOUNT AS OF'
10530052 'THE AUDIT DATE ABOVE.'
10530088 'THIS IS NOT A REQUEST FOR PAYMENT.'
* $Blanks in the output buffer create two blank lines
10540016 'ACCOUNT NUMBER'
10540054 ACCOUNT
10540097 'OUTSTANDING BALANCE'
10540131 BALANCE F2
10540178 'LATE CHARGES DUE'
10540215 ODCHG F2
10540259 'REMAINING PAYMENTS'
10540302 REMAIN F1
10550001 '- ' $New buffer contents 3
10550011 'PLEASE SIGN AND RETURN THIS LETTER IN THE'
10550053 'ENCLOSED POSTAGE'
10550088 'PAID ENVELOPE. IF YOUR RECORDS DO NOT AGREE,'
10550134 'ADDITIONALLY PLEASE'
10550169 'WRITE IN THE CORRECT DATA. YOUR PROMPT REPLY'
10550215 'WILL BE GREATLY'
10550250 'APPRECIATED.'
10560001 '- ' $New buffer contents 4
10560042 'VERY TRULY YOURS,'
10560082 '- ' $Triple space
10560123 'INTERNAL AUDIT DEPARTMENT'
10560163 '- ' $Triple space
10560169 '-----'
10560209 '-----'
10560250 ACCOUNT
10560275 'PLEASE REPLY BELOW'
10570001 '- ' $New buffer contents 5
10570088 '-----'
10570113 '-----'
10570176 'DATE'
10570208 'SIGNATURE'
10570244 '- ' $Triple space
10570250 'COMMENTS -'

```

## Printed Letter

Shown below is a letter which is the result of the code given above.

|                                                                                                                                                                                                                                                                                                                   |                             |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------|
| month dd, yyyy                                                                                                                                                                                                                                                                                                    |                             |
| AMOS      JOHNSON<br>22651 MASS AVENUE<br>SAN FRANCISCO CA 09801                                                                                                                                                                                                                                                  |                             |
| DEAR CUSTOMER:                                                                                                                                                                                                                                                                                                    |                             |
| FROM TIME TO TIME, AS PART OF OUR REGULAR AUDIT PROCEDURE, WE<br>ASK OUR CUSTOMERS TO CONFIRM THAT THEIR RECORDS ARE IN AGREEMENT<br>WITH OURS. THE INFORMATION SHOWN BELOW IS TAKEN FROM OUR RECORDS<br>OF YOUR ***INSTALLMENT LOAN*** ACCOUNT AS OF THE AUDIT DATE ABOVE.<br>THIS IS NOT A REQUEST FOR PAYMENT. |                             |
| ACCOUNT NUMBER<br>OUTSTANDING BALANCE<br>LATE CHARGES DUE<br>REMAINING PAYMENTS                                                                                                                                                                                                                                   | 21056<br>11.23<br>1.00<br>1 |
| PLEASE SIGN AND RETURN THIS LETTER IN THE ENCLOSED POSTAGE<br>PAID ENVELOPE. IF YOUR RECORDS DO NOT AGREE, ADDITIONALLY PLEASE<br>WRITE IN THE CORRECT DATA. YOUR PROMPT REPLY WILL BE GREATLY<br>APPRECIATED.                                                                                                    |                             |
| VERY TRULY YOURS,                                                                                                                                                                                                                                                                                                 |                             |
| INTERNAL AUDIT DEPARTMENT                                                                                                                                                                                                                                                                                         |                             |
| -----<br>21056                                                                                                                                                                                                                                                                                                    | -----<br>PLEASE REPLY BELOW |
| -----<br>DATE                                                                                                                                                                                                                                                                                                     | -----<br>SIGNATURE          |
| COMMENTS -                                                                                                                                                                                                                                                                                                        |                             |

### Printing a Report with Footers

This example produces an Account Gain or Loss report for individual branch offices. A heading, footer, and totals information is printed on each page.

The following code:

- Defines the length of the output buffer as 532 bytes, based on the heading that consists of 4 lines (two with information and two with blanks) and a print lines of 133 characters
- Uses a control break of 1 on the SORT parameter and a page-eject control character ('1') on a type 4 parameter to obtain a page eject when the branch changes
- Suppresses the printing of unused segments of the output buffer by using '\*'
- Specifies 20 lines to a page on the CALL to CULEMLIN. SYS020 is added to the CULE job control statements

- Defines the footer on a type 5 line by using '@'

```
IN 80
REC BRANCH 1 2
REC ACCOUNT 3 3
REC NAME 6 18
REC CURR-BAL 63 7 2 DP=2
REC PREV-BAL 70 7 2 DP=2
140 GAIN-LOSS DP=2
14SORT BRANCH ACCOUNT
14OUT 532 UM(CULEMLIN,20) $20 detail lines per page
14410001 '#' $Heading lines
14410007 '* * CONFIDENTIAL * *'
14410038 'ACCOUNT GAIN OR LOSS'
14410072 '* * CONFIDENTIAL * *'
14410270 'BRANCH'
14410279 'ACCOUNT'
14410298 'NAME'
14410319 'BALANCE'
14410331 'PRIOR BALANCE'
14410349 'GAIN/LOSS'
14420001 '1' $Page eject
```

```

14420134 '*' $Blank buffer nulled out
14420267 '*'
14420400 '*'
14510006 BRANCH
14510013 ACCOUNT
14510023 NAME
14510049 CURR-BAL F2
14510065 PREV-BAL F2
14510081 GAIN-LOSS SZ=11 F2
14510134 '*'
14510267 '*'
14510400 '*'
14520001 '@' $Footer
14520273 '* * CONFIDENTIAL * *'
14520338 '* * CONFIDENTIAL * *'
14520400 '*'
14610001 '- '
14610008 'TOTALS'
14610049 CURR-BAL F2
14610065 PREV-BAL F2
14610081 GAIN-LOSS SZ=11 F2
14610134 '*'
14610267 '*'
14610400 '*'
14620001 '0' $Spacing
14620008 'GRAND TOTALS'
14620049 CURR-BAL F2
14620065 PREV-BAL F2
14620081 GAIN-LOSS SZ=11 F2
14620134 '*'
14620267 '*'
14620400 '*'
147 CURR-BAL MINUS PREV-BAL GAIN-LOSS
148010 IF LEVL EQ 2 100
148 TAKE 1
148100 TAKE 2
140 GAIN-LOSS DP=2

```

| * * CONFIDENTIAL * * |         | ACCOUNT GAIN OR LOSS |           | * * CONFIDENTIAL * * |           |
|----------------------|---------|----------------------|-----------|----------------------|-----------|
| BRANCH               | ACCOUNT | NAME                 | BALANCE   | PRIOR BALANCE        | GAIN/LOSS |
| 15                   | 060     | SHARON ARMSTRONG     | 10,000.00 | 9,100.54             | 899.46    |
| 21                   | 056     | AMOS JOHNSON         | 11.23     | 1,000.01             | 988.78 -  |
| 29                   | 557     | IRWIN TRIMBLE        | 357.85    | 200.02               | 157.83    |
| 30                   | 115     | IRMA DOONES          | 9,756.73  | 340.10               | 9,416.63  |
| 33                   | 470     | VICTORIA DAY         | 50,432.00 | 560.05               | 49,871.95 |
| 69                   | 876     | BRUCE THORPE         | 203.45    | 100.01               | 103.44    |
| 99                   | 083     | HELEN SANTOVEC       | 2,857.43  | 3,450.20             | 592.77 -  |
| TOTALS               |         |                      | 73,618.69 | 14,750.93            | 58,867.76 |
| * * CONFIDENTIAL * * |         | * * CONFIDENTIAL * * |           |                      |           |

## Writing Formatted Records to a VSAM File (CULEVSAM)

### What You Can Do

CULEVSAM writes entry-sequenced (ESDS), key-sequenced (KSDS), and relative-sequenced (RSDS) VSAM records to an already existing VSAM data set.

You can use CULEVSAM to:

- Add ESDS records to the end of the existing VSAM data set
- Add KSDS records according to the ascending value of each key field
- Add RSDS records to null VSAM data set created with the utility IDCAMS expressly for the CULEVSAM run or a previously existing nonblank data set

### How to Use CULEVSAM

Define the VSAM file to which the records are written in the CULE step of the CA Culprit JCL.

| System | JCL statement                         |
|--------|---------------------------------------|
| z/OS   | //SYS020 DD DSN=cluster.name,DISP=SHR |

cluster.name = VSAM file cluster name as defined  
in the IDCAMS jobs that created this file

| System | JCL statement                                                                                                                                                                                                                                                                                           |
|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| z/VSE  | <pre>// ASSGN  SYS020,DISK,VOL=nnnnnn,SHR // DLBL   SYS020,'cluster.name',,VSAM,CAT=IJSYST // EXTENT SYS020,nnnnnn</pre> <p>nnnnnn = volume serial name of disk containing the VSAM data set</p> <p>cluster.name = name of the VSAM cluster as defined in the IDCAMS job that created this data set</p> |
| z/VM   | VSAM files are not supported.                                                                                                                                                                                                                                                                           |

To invoke CULEVSAM:

First—Specify CULEVSAM on an **OUTPUT** parameter using the UM option:

Col

2



RPT-*nn*OUTPUT UM(CULEVSAM , **sequence-type-a**)

- *Rpt-nn*, coded in columns 2 and 3, requires a 2-digit report number in the range 00 through 99.
- *Sequence-type-a*, preceded by a comma, requires the output VSAM file type:

| Use... | For...                                   |
|--------|------------------------------------------|
| ES     | Output to an entry-sequenced VSAM file   |
| KS     | Output to a key-sequenced VSAM file      |
| RS     | Output to a relative-sequenced VSAM file |

Second—Describe the output VSAM records on type 5 parameters.

## Helpful Hints

- When writing records to a KSDS VSAM file, the key field must be in the position originally defined for the VSAM data set.
- KSDS records must be added in ascending order by the key field.

## Segmenting Reports in a VSE/POWER Run (CULEPOWR)

### What You Can Do

CULEPOWR allows specification of print or punch information for one or more reports in CA Culprit run under VSE/POWER. You can use CULEPOWR as:

- An output module to print or punch tasks through CA Culprit's print routines
- A subroutine called by another output module written in code other than CA Culprit (such as, Assembler or COBOL) that has the responsibility for printing or punching tasks

### How to Use CULEPOWR as a CA Culprit Output Module

To invoke CULEPOWR:

First—Specify CULEPOWR on an **OUTPUT** parameter using the UM option:

**Col**  
**2**  
↓  
**RPT-*nn* OUTPUT UM(CULEPOWR) [LP = *lines-per-page-nn*]**

- *Rpt-nn*, coded in columns 2 and 3, requires a 2-digit report number in the range 00 through 99.
- *Lines-per-page-nn* requires a 2-digit number indicating the number of lines to print on a page. The default is 55 lines.

Second—Code **VSE/POWER JECL statements** after the CA Culprit JCL that executes the CULE step:

**Col**  
**1**  
↓  
**RPT-*nn* power-keyword-q JECL-options [comments]**

- *Rpt-nn*, coded in columns 1 and 2, is the number of the CA Culprit report specified on the OUTPUT parameter that requested CULEPOWR.
- *Power-keyword-q*, coded in columns 3 through 5 and enclosed in single quotation marks, requires a LST or PUN keyword, as appropriate.
- Column 6 is a space.
- *JECL-options*, coded in columns 7 through 68, specifies VSE/POWER options required for the run. (Refer to VSE/POWER publications.)
- *Comments*, coded in columns 69 through 80, are optional.

## Helpful Hints

- If an optional RESTART parameter is specified at the end of the CA Culprit JCL, follow RESTART with a /\* parameter before the first VSE/POWER JECL card.  
If no RESTART parameter is specified, precede the first VSE/POWER JECL card with a /\* parameter.
- If CULEPOWR is invoked by more than one report, follow each report's JECL information with a /\* parameter and order the JECL cards by ascending report number.
- To print headings on each page, use the OUTPUT parameter LP= option. Otherwise, the lines-per-page defaults to zero; title and heading lines print only on the first page of the report.
- Invoking CULEPOWR changes the current VSE/POWER JECL options. The new JECL options set by CULEPOWR stay in effect until the end of the CA Culprit job or until another report uses CULEPOWR.

## How to Use CULEPOWR as a Subroutine

To use CULEPOWR as a subroutine, use the name **CULEPWR** and link it with each calling output module, as shown in the following sample job control language:

```
// JOB module-name
// OPTION CATAL
 PHASE module-name,*
 INCLUDE module-name
 INCLUDE CULEPOWR
 ENTRY module-name
// EXEC LNKEDT
/*
```

*Module-name* is the name of the output module that is responsible for all printing or punching.

To invoke CULEPWR from an output module:

First—**Issue a CALL to CULEPWR** from the module:

```
CALL CULEPWR
```

Second—**Include a calling parameter list** that points to the address of the control switch and, if indicated, to the JECL information:

| Use this code... | To indicate that...                                                                                                                            |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------|
| 'C'              | JECL information is passed through the CALL. A second address is expected in the calling parameter list to point to the JECL information area. |
| 'R'              | The JECL information is read through the card reader.                                                                                          |

Third—**Test for the control switch return code value:**

| A return code of... | Means...                                                                                                                               |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| '0'                 | No errors. VSE/POWER segmentation performed.                                                                                           |
| '1'                 | Invalid control switch value found. VSE/POWER segmentation not attempted.                                                              |
| '2'                 | No input card or more than one input card was found in SYSRDR. VSE/POWER segmentation not performed.                                   |
| '3'                 | Internal error.                                                                                                                        |
| '4'                 | Invalid JECL keyword. VSE/POWER segmentation not performed.                                                                            |
| '5'                 | An unrecognizable error code returned after VSE/POWER segmentation was attempted.                                                      |
| '6'                 | A hex '04' error code returned after VSE/POWER segmentation was attempted. See the Segmentation Macro section of the VSE/POWER manual. |
| '7'                 | A hex '08' error code returned after VSE/POWER segmentation was attempted. See the Segmentation Macro section of the VSE/POWER manual. |

## Helpful Hints

- VSE/POWER JECL information is supplied through the CALL or through the card reader.
- CULEPOWR accesses JECL information through a control switch. The control switch is pointed to by the first address in the calling parameter list. The value of the control switch determines if a second address should be expected in the parameter list. If present, the second address points to the JECL information.
- The JECL information area has the following format:

Col

1



**power-keyword power-JECL-options**

- *Power-keyword*, coded in columns 1 through 3 and enclosed in single quotation marks, requires a LST or PUN keyword, as appropriate.
- Column 4 is a space.
- *Power-JECL-options*, coded in columns 5 through 66, specifies VSE/POWER options required for the run. (Refer to VSE/POWER publications.)

- The VSE/POWER JECL information card, read through a CALL to CULEPOWR, has the following format:

**Col**

**3**



***power-keyword power-JECL-options [ comments ]***

Columns 1 and 2 are not used.

*Power-keyword*, coded in columns 3 through 5, requires a LST or PUN keyword, as appropriate.

Column 6 is a space.

*Power-JECL-options*, coded in columns 7 through 68, specifies VSE/POWER options required for the run. (Refer to VSE/POWER publications.)

*Comments*, coded in columns 69 through 80, are optional.

- When the JECL card is read through a CALL to CULEPOWR, the CA Culprit report number is not available for verification against the card. The input job stream of JECL cards must be in ascending numerical sequence of the reports that require them as input.
- Invoking CULEPOWR changes the current VSE/POWER JECL options. The new JECL options set by CULEPOWR stay in effect until the end of the CA Culprit job or until another report uses CULEPOWR.

### Example 1—Calling CULEPWR from an Assembler Routine

#### Reading JECL Information from a Card

```

 .
 .
 .
 CALL CULEPWR, (CTLSW) CALL TO CULEPWR
 CLI CTLSW,X'F0' CHECK RETURN CODE
 BNE ERROR ERROR WAS NOTED
 .
 .
 .
CTLSW DS CLI'R' JECL VIA CARD

```

#### Passing JECL Information Via the Call

```

 .
 .
 .
 CALL CULEPWR, (CTLSW,JECL) CALL TO CULEPWR
 CLI CTLSW,X'F0' CHECK RETURN CODE
 BNE ERROR ERROR WAS NOTED
 .
 .
 .
CTLSW DS CLI'C' JECL VIA CALL
JECL DS CL66'LST FNO=ACB,DISP=H,PRI=1'

```

**Example 2—Calling CULEPWR from a COBOL Module****Reading JECL Information from a Card**

```
WORKING-STORAGE SECTION.
01 CULEPWR-CALL.
 05 CONTROL-SWR PIC X(1) VALUE 'R'.
 05 CONTROL-SWC PIC X(1) VALUE 'C'.
 05 JECL-INFO.
 10 JECL PIC X(24)
 VALUE 'LST FNO=ACB,DISP=H,PRI=1'.
 10 FILLER PIC X(42) VALUE SPACES.
PROCEDURE DIVISION.
.
.
.
CULEPWR-READING-JECL-CARD.
 CALL CULEPWR USING CONTROL-SWR.
 IF CONTROL-SWR NOT EQUAL TO '0'
 GO TO CULEPWR-ERROR.
.
.
.
CULEPWR-JECL-VIA-CALL.
 CALL CULEPWR USING CONTROL-SWC JECL INFOR.
 IF CONTROL-SWC NOT EQUAL TO '0'
 GO TO CULEPWR-ERROR.
```



# Chapter 5: Writing User Modules

---

This section contains the following topics:

[What You Can Do](#) (see page 145)

[General Considerations for User-written Modules](#) (see page 146)

[How to Link-edit User Modules](#) (see page 146)

[How to Write Input Modules](#) (see page 150)

[How to Write Procedure Modules](#) (see page 161)

[How to Write Output Modules](#) (see page 165)

## What You Can Do

You can write your own user modules to facilitate processing CA Culprit reports. Before writing your own module, be sure to check the CA-supplied modules to see if the function you need is already available. The CA-supplied modules are listed in "Introduction" chapter.

### Typical Uses for User-written User Modules

Input, procedure, and output modules perform specific processing tasks that have practical applications. Some typical uses are listed below:

| Use...             | To...                                                          |
|--------------------|----------------------------------------------------------------|
| An input module    | Decompress a file                                              |
|                    | Read a file type not supported by CA Culprit                   |
|                    | Combine records from different files to form one input buffer  |
| A procedure module | Perform a difficult computation                                |
|                    | Incorporate a company-required routine                         |
| An output module   | Write a file or report in a format not available to CA Culprit |
|                    | Compress the output                                            |

## General Considerations for User-written Modules

- User-written modules can be called at the same points during a CA Culprit run as the CA-supplied modules discussed in this manual. These points occur during:
  - **Input file processing** (the extract phase), called from the UM option of the INPUT parameter:  

```
INPUT 80 F 400 UM(module)
```
  - **Input record processing** (the extract phase), called from type 7 logic:  

```
017010 CALL module
```
  - **Output processing** (the output phase), called from the OUTPUT parameter:  

```
010OUT UM(module)
```
- Modules are treated as subroutines by the main CA Culprit processing logic, as follows:
  - **The module executes** at a specified exit point.
  - **CA Culprit passes data**, in the form of an argument list, to the module.
  - **Control is returned to the CA Culprit program** when the module finishes executing.
- Modules must be written in a programming language that observes standard linkage conventions, such as COBOL, Assembler, PL/I or FORTRAN.
- All modules must be compiled and link edited.

If a module already exists in a load (core-image) library, no new link is required. You can do one of the following:

  - Concatenate the library containing the module with the CA Culprit load (core-image) library.
  - Copy the library member into the CA Culprit load (core-image) library.
- When procedure modules are link edited with names in the CULLUSnn format, care must be taken that the name does not duplicate a CA-supplied module.

## How to Link-edit User Modules

Under all operating systems each module is link edited separately. The member name of a user module within the load library must be the same name used in the CA Culprit code to call the module.

Under z/VSE, the space between the end of PHASE CULLGEN and the beginning of PHASE CULLWORK should be as large as the environment permits. Procedure modules for all the reports in a single run must be stored together.

## Establishing Linkage to a COBOL Module

Because CA Culprit system programs are written in the Assembler, each invocation of a COBOL program is normally treated as an entry into a main program. Special procedures must be used in order for the COBOL program to act as a subroutine. This is necessary, for example, if a file is opened on the first call to the COBOL program and that file is accessed in subsequent calls.

Use either of two methods shown below to establish linkage to a COBOL procedure module. Use method 2 to establish linkage to a COBOL input or output module.

**METHOD 1:** Call CULLUS00 as documented in The Universal Interface (CULLUS00). CULLUS00 will automatically create the COBOL environment before calling the target procedure module. Since CULLUS00 is only used for calling procedure modules, this method will not work for input or output modules.

If this method is used, the target procedure module can be linked with any desired AMODE or RMODE under z/OS, z/VM or z/VSE operating systems. If running under IBM Language Environment, the procedure module must be linked AMODE(31).

**METHOD 2:** Calling the COBOL module directly. This method can be used with modules compiled under VS COBOL, VS COBOL II, LE-compliant COBOL compilers (such as COBOL for and z/VM or COBOL/VSE) COBOL. To implement this method, perform the following steps:

1. Use the compile options listed below (if they apply to your version of the COBOL compiler):
  - NOENDJOB
  - NODYNAM
2. Code the verb GOBACK within the module code.

3. For LE-compliant COBOL compilers under VSE such as COBOL/VSE, add the following statement to the linkage editor control statements:

- INCLUDE CEEUOPT

where CEEUOPT names a file that contains a CEEUOPT module which was compiled specifying RTEREUS=YES.

**Note:**

- For more information on creating an application-specific version of CEEUOPT, see the appropriate COBOL Application Programming Guide.
- It is not necessary to link a CEEUOPT module specifying RTEREUSE=YES with COBOL programs compiled with Enterprise COBOL for z/OS.

4. For COBOL II, compile and add the following statement to the linkage editor control statements:

- INCLUDE IGZEOPT

where IGZEOPT names a file which contains an IGZEOPT module which was compiled specifying RTEREUS=YES. See the VS COBOL II Application Programming Guide for information on creating an application-specific version of IGZEOPT.

5. Specify any desired AMODE, but specify RMODE(24).

## Establishing Linkage to an Assembler Module

When a user module is written in Assembler, observe the following linking conventions:

- Use an ENTRY statement if the entry point of the module is not the beginning of the first control section.
- Use standard register assignments:

| Register... | Assigns...                              |
|-------------|-----------------------------------------|
| 1           | The address of the argument list        |
| 2-12        | Must be saved and restored              |
| 13          | The address of the SAVE area            |
| 14          | The return address in CA Culprit        |
| 15          | The entry address of the CALLED program |

- Assemble and link the module into a load (z/OS) or core-image (z/VSE) library.

## Establishing Linkage to a PL/I Module

Because the operating environment of a PL/I program is different from that of CA Culprit (that is, Assembler), the following procedures must be observed when using PL/I modules.

**Note:** PL/I user modules are only supported on the z/OS platform.

For programs compiled with an LE-compliant compiler:

- Write the module without the MAIN option but the FETCHABLE option.
- Use CULLEOPT, a CA-supplied module, to establish the PL/I environment.
- Assemble and link the module into a load library.
- Relink the module, including CULLEOPT in the link edit. The linkage editor control statements for relinking CULLEOPT (in the load library) and an already compiled and linked PL/I user module (plisub) in a z/OS system are shown below:

```
//SYSIN DD *
CHANGE CULSUB(plisub_proc) Procedure name within PL/I user module
INCLUDE SYSLMOD(plisub) Linked PL/I user module name
INCLUDE SYSLMOD(CULLEOPT)
ENTRY CULLEOPT
NAME CULLUSnn(R) Name called by CA Culprit
/*
```

For programs compiled with a non LE-compliant compiler:

- Write the module without the MAIN option.
- Use CULLPOPT, a CA-supplied module, to establish the PL/I environment.
- Assemble and link the module into a load library.
- Relink the module, including CULLPOPT in the link edit. The linkage editor control statements for relinking CULLPOPT (in the load library) and an already compiled and linked PL/I user module (plisub) in a z/OS system are shown below:

```
//SYSIN DD*
CHANGE CULPLI(plisub_proc) Procedure name within PL/I user module
INCLUDE SYSLMOD(plisub) Linked PL/I user module name
INCLUDE SYSLMOD(CULLPOPT)
ENTRY PLIOPT
NAME CULLUSnn(R) Name called by CA Culprit
/*
```

## Establishing linkage to a FORTRAN module

The differences between a FORTRAN module and the CA Culprit program (that is, Assembler) require:

- Using the naming convention CULFUS $nn$ , where  $nn$  is a 2-digit number.
- Using the CA-supplied module CULLUS00 to call the module from the type 7 logic in the main CA Culprit program. CULLUS00 recognizes the CULFUS $nn$  naming convention and automatically converts the FORTRAN numeric fields. (See The Universal Interface (CULLUS00) for more information about CULLUS00.)

## How to Write Input Modules

### What You Can Do

A standard CA Culprit module reads one record at a time from the input file and places that record in the input buffer. When using an input module, you can:

- Open and close an input file or set of files
- Read an input file or a set of files into the CA Culprit input buffer
- Check for errors on the input file
- Perform computations and record formatting before the input buffer is built
- Process selected records when SELECT/BYPASS parameters are encountered

### How Information Is Passed

Information is passed between CA Culprit and an input module by means of an **argument list**, which is set up by CA Culprit from system and user-supplied information. The following considerations apply:

- Most of the arguments are not accessed by the input module.
- References to the arguments can occur only within the input module code.
- Arguments 1 (input buffer address) and 3 (open/closeswitch) must always be accessed within the input module code.

The input module arguments, their function, and address pointers are listed in the following table.

### Input Module Argument List and Address Pointers

Addresses are passed by the CULL step through Register 1 to all input modules called in a CA Culprit run.

| Argument name | Function/comments                                                                                                                                 | Displacement from Register 1 |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------|
| INPUT         | The address of the CA Culprit input buffer accessed within the input module code.                                                                 | 0                            |
| DEVINDS       | The address of type and DTF codes used internally by CA Culprit.                                                                                  | 4                            |
| OPNCLS        | The address of a 1-byte user-supplied open/close switch accessed within the input module code. The following table lists allowable switch values. | 8                            |
| FILSPEC       | The address for record and label-type codes for the input file being read.                                                                        | 12                           |
| RECSIZE       | The address for the record size (halfword) of the input file records.                                                                             | 16                           |
| BLKSIZE       | The address for the blocksize (halfword) of the input file.                                                                                       | 20                           |
| FILDESC       | The address for the external file name (8-bytes alphanumeric) and logical unit (1-byte hexadecimal) for the input file.                           | 24                           |
| ISKEY         | The address of an internal CA Culprit key field not accessible by the user.                                                                       | 28                           |

| Argument name | Function/comments                                                                                                                                                                                                                                                                                                         | Displacement from Register 1 |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------|
| VPRINT        | <p>The entry address for the CA Culprit print routine used to print user-defined diagnostic and error message relating to the input buffer.</p> <p>VPRINT cannot be called by a COBOL user module. For Assembler modules, call the VCON print routine address and pass the address of the line of data to be printed.</p> | 32                           |
| ASELTBL       | The address of the internal select table used by input module select routines (CULLCBSL, CULSLCT, and CULSINIT).                                                                                                                                                                                                          | 36                           |
| VSEL          | The VCON address for the CULLSEL module used by input module select routines.                                                                                                                                                                                                                                             | 40                           |
| VDBEXIT       | The address for the DBEXIT module supplied by the input module for the CULL step. VDBEXIT is not accessible by the user.                                                                                                                                                                                                  | 44                           |
| COMMONA       | A common area for internal CA Culprit use. COMMONA is not accessible by the user.                                                                                                                                                                                                                                         | 48                           |
| ALTDESC       | The address if the filename/ddname and logical unit for an alternate file specified with the INPUT parameter DD2 option.                                                                                                                                                                                                  | 52                           |
| PASSWD        | The address of an 8-byte user-specified password, as specified on the INPUT parameter.                                                                                                                                                                                                                                    | 56                           |

**Input Module Open/close Switch Values**

If the open/close switch is not set to one of these values, CA Culprit outputs an error message stating the contents of the switch.

| Hexadecimal value | File status                                                                                                                                                                                                     | Binary value |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------|
| X'FF'             | The input file is closed and must be opened by resetting the value to X'00'.                                                                                                                                    | 255          |
| X'00'             | The input file is open and can read and deliver records to the CA Culprit input buffer until an end-of-file condition is encountered. Close the file and reset this value to X'FF' when end-of-file is reached. | 0            |
| X'0F'             | The STOP action has been encountered in procedure code. The input module closes the input file. Reset this value to X'FF'.                                                                                      | 15           |
| X'F0'             | The input module code can use this value to indicate an I/O error. If used and an I/O error is found, reset this value to X'FF'.                                                                                | 240          |

## Coding a COBOL Input Module

### How to Implement SELECT/BYPASS Logic

CA supplies a COBOL subroutine (CULLCBSL) to select input records for processing. When SELECT/BYPASS parameters are entered in CA Culprit code for any run that calls the input module, the CA routines must be called from the module to implement the selection logic.

To implement SELECT/BYPASS logic:

- **Include CULLCBSL** in the link of the input module. You can find CULLCBSL in the z/OS CAGJLOAD library that is created during installation or in the z/VSE IDMS sublibrary. If the input module is compiled with an Enterprise COBOL compiler, link the module with a CULLCBSL program (for CULPRIT releases 18.0 and higher).

- **Call CULLCBSL** from the COBOL input module:

CALL 'CULLCBSL' USING *input-buffer-addr select-switch-addr*

- *Input-buffer-addr*, requires the name of a field that points to the starting address of the CA Culprit input buffer.
- *Select-switch-addr*, requires the name of a 1-byte field to hold file SELECT/BYPASS codes.

### Example

This is an example of a COBOL input module that reads an 80-byte record.

- The fields referenced by the argument list that CA Culprit passes to the module are defined in the LINKAGE SECTION of the DATA DIVISION.
- The USING clause of the PROCEDURE DIVISION statement lists the data names assigned in the LINKAGE SECTION that are used by the subroutine. References to the data names serve as direct substitutes for the storage addresses.

### Input Module

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CULLUS98.
INSTALLATION. COMPUTER ASSOCIATES
DATE-WRITTEN. MONTH YYYY.
REMARKS. THIS IS A TEST OF A COBOL USER INPUT MODULE
 FOR A CULPRIT JOB.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-370.
OBJECT-COMPUTER. IBM-370.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
 SELECT INPUT-FILE ASSIGN TO UT-S-SYS010.
*CHANGE FILE ASSIGNMENT FOR VSE/ESA TO: SYS010-UT-S.
```

```
DATA DIVISION.
FILE SECTION.
FD INPUT-FILE
 RECORDING MODE IS F
 LABEL RECORDS ARE STANDARD
 RECORD CONTAINS 80 CHARACTERS
 BLOCK CONTAINS 0 RECORDS
 DATA RECORD IS RECORD-IN.
01 RECORD-IN PIC X(80).
WORKING-STORAGE SECTION.
77 SEL-BYP-SW PIC X.
77 CLOSE-STATUS PIC X VALUE ' '.
77 OPEN-STATUS PIC X VALUE ' '.
* CLOSE-STATUS = HEX 'FF' OPEN-STATUS = HEX '00'
LINKAGE SECTION.
01 CULARG-INPUT PIC X(80).
01 CULARG-2 PIC X.
01 CULARG-SW PIC X.
01 CULARG-3 PIC XX.
01 CULARG-4 COMP PIC 99.
01 CULARG-5 COMP PIC 99.
01 CULARG-6 PIC X(8).
01 CULARG-7 PIC X.
01 CULARG-8 PIC X.
```

```
PROCEDURE DIVISION USING CULARG-INPUT
 CULARG-2
 CULARG-SW
 CULARG-3
 CULARG-4
 CULARG-5
 CULARG-6
 CULARG-7
 CULARG-8.

0010-CONTROL.
 MOVE ' ' TO SEL-BYP-SW.
 IF CULARG-SW = CLOSE-STATUS
 PERFORM 0020-OPEN THRU 0020-EXIT
 ELSE
 IF CULARG-SW = OPEN-STATUS
 PERFORM 0030-READ THRU 0030-EXIT
 UNTIL SEL-BYP-SW = 'Y'
 ELSE
 PERFORM 0040-CLOSE THRU 0040-EXIT.
 GOBACK.

0020-OPEN.
 OPEN INPUT INPUT-FILE.
 MOVE OPEN-STATUS TO CULARG-SW.
 PERFORM 0030-READ THRU 0030-EXIT
 UNTIL SEL-BYP-SW = 'Y'.

0020-EXIT.
 EXIT.

0030-READ.
 READ INPUT-FILE INTO CULARG-INPUT
 AT END PERFORM 0040-CLOSE THRU 0040-EXIT
 MOVE CLOSE-STATUS TO CULARG-SW
 MOVE 'Y' TO SEL-BYP-SW
 GO TO 0030-EXIT.
 MOVE 'Y' TO SEL-BYP-SW.
 CALL 'CULLCBSL' USING CULARG-INPUT
 SEL-SYP-SW.

0030-EXIT.
 EXIT.

0040-CLOSE.
 CLOSE INPUT-FILE.

0040-EXIT.
 EXIT.
```

## Coding an Assembler Input Module

### How to Implement SELECT/BYPASS Logic

SELECT/BYPASS logic can be implemented in an Assembler input module by using the CA-supplied CULSINIT and CULSLCT macros in the source library from the install. CULSINIT establishes an environment that allows CA Culprit SELECT/BYPASS logic to function. CULSLCT implements the CA Culprit SELECT/BYPASS logic.

To implement SELECT/BYPASS logic:

1. **Code CULSINIT** immediately after the initial register and save area housekeeping functions in the input module source code.
2. **Code CULSLCT** in the input module logic.

Register 1 points to the input record to which SELECT/BYPASS logic is applied. Use these two positional operands:

- The label of the location receiving control if the record passes the SELECT/BYPASS logic. Absence of SELECT/BYPASS parameters in the CA Culprit code causes control to be passed to this location after each use of the CULSLCT macro.
- The label of the location receiving control if the record fails the SELECT/BYPASS logic.

### Example

This is an example of an Assembler input module that reads an 80-byte record.

#### Input Module

```
CULLUS95 CSECT
R0 EQU 0
R1 EQU 1
R2 EQU 2
R3 EQU 3
R4 EQU 4
R5 EQU 5
R6 EQU 6
R7 EQU 7
R8 EQU 8
R9 EQU 9
R10 EQU 10
R11 EQU 11
R12 EQU 12
```

```

R13 EQU 13
R14 EQU 14
 STM 14,12,12(13) SAVE CALLER'S REGISTERS
 BALR R3,0 ESTABLISH BASE REGISTER
 USING *,R3
 ST R13,SAVEAREA+4
 LA R13,SAVEAREA
 ST 13,SAVEAREA+8
 CULSINIT ESTABLISHES CULPRIT ENVIRONMENT
* FOR SELECT/BYPSS CALLS
 L R4,0(R1) R4 POINTS TO INPUT BUFFER
 L R5,8(R1) R5 POINTS TO OPEN/CLOSE SW
 CLI 0(R5),X'FF' IS FILE CLOSED?
 BE OPEN YES
 CLI 0(R5),X'00' IS FILE OPEN?
 BE READ YES
 B CLOSE

RETURN EQU *
 L R13,SAVEAREA+4 RESTORE REGISTERS
 LM R14,R12,12(R13)
 BR R14
SAVEAREA DS 18F SAVE REGISTER AREA
OPEN EQU *
 OPEN (INPUTFI,INPUT) OPEN INPUT FILE
 MVI 0(R5),X'00' SET OPEN/CLOSE SW TO OPEN
READ EQU *
 GET INPUTFI GET FIRST RECORD
 MVC 0(80,R4),0(R1) MOVE FIRST RECORD INTO BUFFER
 CULSLCT RETURN,READ SEL/BYP MACRO
 B RETURN
EOF EQU *
 MVI 0(R5),X'FF' SET OPEN/CLOSE SW TO CLOSE
CLOSE EQU *
 CLOSE INPUTFI CLOSE INPUT FILE
 B RETURN
INPUTFI DCB DSORG=PS,MACRF=GL,DDNAME=SYS010,EODAD=EOF CHANGE FOR VSE/ESA
 END CULLUS95

```

## Coding a PL/I Input Module

### How to Implement SELECT/BYPASS Logic

Because the PL/I environment is different from CA Culprit (that is, Assembler), SELECT/BYPASS logic must be done with the SELECT/BYPASS BUFFER statement within the CA Culprit program.

### Example

This is an example of a PL/I input module input that reads an 80-byte record.

### Input Module

```
PLIPROG:PROC(BUF,ARG1,ARG2,ARG3,ARG4,ARG5,ARG6,ARG7,ARG8);
DCL (BUF, ARG1, ARG2, ARG3, ARG4, ARG5, ARG6, ARG7, ARG8) FIXED(1);
DCL REC CHAR(80) BASED(P1);
DCL FLAG CHAR(1) BASED(P3);
DCL (P1, P2, P3, P4, P5, P6, P7, P8) POINTER;
DCL ADDR BUILTIN;
DCL SW1 CHAR(1) INITIAL(' '); /* HEX 00*/;
DCL SW2 CHAR(1) INITIAL(' '); /* HEX FF*/;
DCL SYS010 FILE INPUT RECORD;
ON ENDFILE(SYS010) GO TO EOF;
P1 = ADDR(BUF);
P3 = ADDR(ARG2);
IF FLAG = SW2 THEN DO;
 FLAG=SW1;
 OPEN FILE(SYS010);
 END;
IF FLAG = SW1 THEN DO;
 READ FILE(SYS010) INTO (REC);
 GO TO GO_BACK;
 END;
EOF:CLOSE FILE(SYS010);
 FLAG=SW2;
GO_BACK:RETURN;
END PLIPROG;
```

## How to Write Procedure Modules

### What You Can Do

Up to 100 procedure modules can be called from type 7 logic for each report in a CA Culprit run. When you use a procedure module, you can:

- Open and close any file other than the CA Culprit input file
- Read and write any file other than the input file
- Perform customized or complicated procedural routines and return to the statement in type 7 logic following the CALL to the procedure module

### How Information Is Passed

Communication between CA Culprit and a procedure module is effected by an argument list that is set up by CA Culprit and accessed by the module. Before calling a procedure module, CA Culprit supplies a list of one fixed argument and nine user-supplied arguments that consist of system and user-supplied information:

- The **fixed argument** is the starting address of the CA Culprit input buffer, which makes it possible for any field in the input buffer to be accessed by the procedure module.
- The **user-supplied arguments** can point to numeric or alphanumeric work fields, literals, numeric constants, or individual fields from the input record. Once the address of a field is passed to the procedure module, the contents of the field can then be processed or modified by the module.

## Coding a COBOL Procedure Module

The following module adds two numbers and returns the result to the main CA Culprit program.

### Procedure Module

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CULLUS97.
INSTALLATION. COMPUTER ASSOCIATES
DATE-WRITTEN. MONTH YYYY.
REMARKS. THIS IS A TEST OF A COBOL PROCEDURE MODULE
 FOR A CULPRIT JOB.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-370.
OBJECT-COMPUTER. IBM-370.
DATA DIVISION.
LINKAGE SECTION.
01 CULARG-INPUT PIC X.
01 CULARG-1 PIC S9(15) COMP-3.
01 CULARG-2 PIC S9(15) COMP-3.
01 CULARG-3 PIC S9(15) COMP-3.
PROCEDURE DIVISION USING CULARG-INPUT
 CULARG-1
 CULARG-2
 CULARG-3.

0010-CONTROL.
 ADD CULARG-1 CULARG-2 GIVING CULARG-3.
 GOBACK.
```

## Coding an Assembler procedure module

The following module adds two numbers and returns the result to the main CA Culprit program.

### Procedure Module

```

CULLUS94 CSECT
R0 EQU 0
R1 EQU 1
R2 EQU 2
R3 EQU 3
R4 EQU 4
R5 EQU 5
R6 EQU 6
R7 EQU 7
R8 EQU 8
R9 EQU 9
R10 EQU 10
R11 EQU 11
R12 EQU 12
R13 EQU 13

R14 EQU 14
R15 EQU 15
 STM R14,R12,12(R13) SAVE CALLER'S REGISTERS
 BALR R3,0 ESTABLISH BASE REGISTER
 USING *,R3
 ST R13,SAVEAREA+4
 LA R13,SAVEAREA
 ST R13,SAVEAREA+8
PROCESS EQU *
 LM R4,R6,4(R1) R4 → R6 POINTS TO ARG1 → ARG3
 ZAP 0(8,R6),0(8,R4) ZERO ARG3, ADD ARG1
 AP 0(8,R6),0(8,R5) ADD ARG2
RETURN EQU *
 L R13,SAVEAREA+4 RESTORE REGISTERS
 LM R14,R12,12(R13)
 BR R14
SAVEAREA DS 18F SAVE REGISTER AREA
 B RETURN
 END CULLUS94

```

## Coding a PL/I Procedure Module

The following module adds two numbers and returns the result to the main CA Culprit program.

### Procedure Module

```
PLIPROG:PROC(BUF,ARG1,ARG2,ARG3);
DCL (BUF, ARG1, ARG2, ARG3) FIXED(1);
DCL REC CHAR(80) BASED(P1);
DCL NUMERIC1 FIXED DEC(15) BASED (P2);
DCL NUMERIC2 FIXED DEC(15) BASED (P3);
DCL NUMERIC3 FIXED DEC(15) BASED (P4);
DCL (P1, P2, P3, P4) POINTER;
DCL ADDR BUILTIN;
P1 = ADDR(BUF);
P2 = ADDR(ARG1);
P3 = ADDR(ARG2);
P4 = ADDR(ARG3);
NUMERIC3=NUMERIC1 + NUMERIC2;
RETURN;
END PLIPROG;
```

## Coding a FORTRAN procedure module

The following module adds two numbers and returns the result to the main CA Culprit program.

### The procedure module

```
INTEGER*4 A,B,C
C=A+B
RETURN
END
```

## Helpful hints

- To call a FORTRAN module in a CA Culprit run, you must use CULLUS00. CULLUS00 converts the CA Culprit 8-byte and 16-byte packed decimal work fields, which are not recognized by FORTRAN, to double precision floating point numeric values.
- FORTRAN modules must use the CULFUS $nn$  naming convention.

## How to Write Output Modules

### What You Can Do

An output module serves as a bridge between the output record built by CA Culprit and the actual output from the CA Culprit run. If you cannot find the format for a file or report in the CA-supplied output modules, you can write your own output module.

### How Information Is Passed

Communication between CA Culprit and an output module occurs through an argument list that is set up by CA Culprit and accessed by the output module. Before calling an output module, CA Culprit supplies the argument list values from system and user-supplied information. The following considerations apply:

- Most of the arguments are not accessed by the output module.
- References to the arguments can occur only within the output module code.
- Arguments 1 (output record address) and 3 (open/close switch) must always be accessed within the output module code.

The following table lists the output module arguments, their function, and address pointers.

#### Output Module Argument List and Address Pointers

Addresses are passed by the output phase through Register 1 to all output modules called in a CA Culprit run.

| Argument Name | Function/Comments                                                                                                                                                                      | Displacement from Register 1 |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------|
| OUTBUF        | The starting address of the CA Culprit output record, containing records formatted according to user specifications.                                                                   | 0                            |
| OUTCODS       | The address of a field containing 2 bytes of internal CA Culprit code (device type and DTF code) and 2 bytes for the output report number. The report number is accessible for output. | 4                            |
| OUTOCLS       | The address of a 1-byte user-supplied open/close switch accessed within the output module code. The following table lists allowable switch values.                                     | 8                            |

| Argument Name | Function/Comments                                                                                                                                                                                                                                                                                                   | Displacement from Register 1 |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------|
| OUTSPEC       | The address of the record type code for the output file being written (2 bytes) and 2-bytes unused space.                                                                                                                                                                                                           | 12                           |
| OUTREC        | The address for the size (4-bytes) of the output file records.                                                                                                                                                                                                                                                      | 16                           |
| OUTBLK        | The address for the output file blocksize (4 bytes).                                                                                                                                                                                                                                                                | 20                           |
| OUTDESC       | The address for the external file name (8-bytes alphanumeric) and logical unit (1-byte hexadecimal) for the output file.                                                                                                                                                                                            | 24                           |
| OUTISK        | The address of an internal CA Culprit key field not accessible by the user.                                                                                                                                                                                                                                         | 28                           |
| OUTPRINT      | The entry address for the CA Culprit print routine used to print user-defined diagnostic and error message relating to the output buffer.<br><br>VPRINT cannot be called by a COBOL user module. For Assembler modules, call the VCON print routine address and pass the address of the line of data to be printed. | 32                           |
| OUTFORM       | The address of a 10-byte field containing 2 bytes that contain the special value that is coded on the OUTPUT parameter.                                                                                                                                                                                             | 36                           |

#### Output Module Open/close Switch Values

If the open/close switch is not set to one of these values, CA Culprit outputs an error message stating the contents of the switch.

| Hexadecimal Value | Binary Value                                                                                                                                                | File Status |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| X'FF'             | The output file is closed and must be opened by resetting the value to X'00'.                                                                               | 255         |
| X'00'             | The output file is open and reads records passed from CA Culprit and writes them to an output device or file until an end-of-file condition is encountered. | 0           |

| Hexadecimal Value | Binary Value                                                                                                                              | File Status |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| X'0F'             | Indicates end-of-file after the last output record is delivered to the output module. The file is closed and the value is reset to X'FF'. | 15          |

## Coding a COBOL Output Module

The following example writes an 80-byte record.

### Output Module

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CULLUS96.
INSTALLATION. COMPUTER ASSOCIATES
DATE-WRITTEN. MONTH YYYY.
REMARKS. THIS IS A TEST OF A COBOL OUTPUT MODULE
 FOR A CULPRIT JOB.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-370.
OBJECT-COMPUTER. IBM-370.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
 SELECT OUTPUT-FILE ASSIGN TO UT-S-SYS004.

* USE SYS004-UT-S FOR z/VSE

DATA DIVISION.
FILE SECTION.
FD OUTPUT-FILE
 RECORDING MODE IS F
 LABEL RECORDS ARE STANDARD
 RECORD CONTAINS 80 CHARACTERS
 BLOCK CONTAINS 0 RECORDS
 DATA RECORD IS RECORD-OUT.
01 RECORD-OUT PIC X(80).
WORKING-STORAGE SECTION.
77 CLOSE-STATUS PIC X VALUE ' '.
77 OPEN-STATUS PIC X VALUE ' '.
* CLOSE-STATUS = HEX 'FF' OPEN-STATUS = HEX '00'
LINKAGE SECTION.
01 CULARG-INPUT PIC X(80).
01 CULARG-2 PIC X.
01 CULARG-SW PIC X.
01 CULARG-3 PIC XX.
01 CULARG-4 COMP PIC 99.
01 CULARG-5 COMP PIC 99.
01 CULARG-6 PIC X(8).
01 CULARG-7 PIC X.
01 CULARG-8 PIC X.
```

```
PROCEDURE DIVISION USING CULARG-INPUT
 CULARG-2
 CULARG-SW
 CULARG-3
 CULARG-4
 CULARG-5
 CULARG-6
 CULARG-7
 CULARG-8.

0010-CONTROL.
 IF CULARG-SW = CLOSE-STATUS
 PERFORM 0020-OPEN THRU 0020-EXIT
 ELSE
 IF CULARG-SW = OPEN-STATUS
 PERFORM 0030-WRITE THRU 0030-EXIT
 ELSE
 PERFORM 0040-CLOSE THRU 0040-EXIT.
 GOBACK.

0020-OPEN.
 OPEN OUTPUT OUTPUT-FILE.
 MOVE OPEN-STATUS TO CULARG-SW.
 PERFORM 0030-WRITE THRU 0030-EXIT
0020-EXIT.
 EXIT.

0030-WRITE.
 WRITE RECORD-OUT FROM CULARG-OUTPUT.
0030-EXIT.
 EXIT.

0040-CLOSE.
 CLOSE OUTPUT-FILE.
 MOVE CLOSE-STATUS TO CULARG-SW.
0040-EXIT.
 EXIT.
```

## Coding an Assembler Output Module

The following example writes an 80-byte record.

### Output Module

```
CULLUS93 CSECT
R0 EQU 0
R1 EQU 1
R2 EQU 2
R3 EQU 3
R4 EQU 4
R5 EQU 5
R6 EQU 6
R7 EQU 7
R8 EQU 8
R9 EQU 9
R10 EQU 10
R11 EQU 11
R12 EQU 12
R13 EQU 13
R14 EQU 14
R15 EQU 15
 STM R14,R12,12(R13) SAVE CALLER'S REGISTERS
 BALR R3,0 ESTABLISH BASE REGISTER
 USING *,R3
 ST R13,SAVEAREA+4
 LA R13,SAVEAREA
 ST R13,SAVEAREA+8
 L R4,0(R1) R4 POINTS TO OUTPUT BUFFER
 L R5,8(R1) R5 POINTS TO OPEN/CLOSE SW
 CLI 0(R5),X'FF' IS FILE CLOSED?
 BE OPEN YES
 CLI 0(R5),X'00' IS FILE OPEN?
 BE WRITE YES
 B CLOSE
RETURN EQU *
 L R13,SAVEAREA+4 RESTORE REGISTERS
 LM R14,R12,12(R13)
 BR R14
```

```

SAVEAREA DS 18F SAVE REGISTER AREA
OPEN EQU *
 OPEN (OUTPUTFI,OUTPUT) OPEN OUTPUT FILE
 MVI 0(R5),X'00' SET OPEN/CLOSE SW TO OPEN
WRITE EQU *
 MVC OUTAREA,0(R4) MOVE OUTPUT BUFFER TO OUTAREA
 PUT OUTPUTFI,OUTAREA WRITE RECORD
 B RETURN
CLOSE EQU *
 CLOSE OUTPUTFI CLOSE OUTPUT FILE
 MVI 0(R5),X'FF' SET OPEN/CLOSE SW TO CLOSE
 B RETURN
OUTAREA DS CL80 OUTPUT RECORD
OUTPUTFI DCB DSORG=PS,MACRF=PM,DDNAME=SYS004,RECFM=FBA,BLKSIZE=80, X
 LRECL=80
 END CULLUS93

```

## Coding a PL/I Output Module

The following example writes an 80-byte record.

### Output Module

```

PLIPROG:PROC(BUF,ARG1,ARG2,ARG3,ARG4,ARG5,ARG6,ARG7,ARG8);
DCL (BUF, ARG1, ARG2, ARG3, ARG4, ARG5, ARG6, ARG7, ARG8) FIXED(1);
DCL REC CHAR(80) BASED(P1);
DCL FLAG CHAR(1) BASED(P3);
DCL (P1, P2, P3, P4, P5, P6, P7, P8) POINTER;
DCL ADDR BUILTIN;
DCL SW1 CHAR(1) INITIAL(' '); /* HEX 00*/;
DCL SW2 CHAR(1) INITIAL(' '); /* HEX FF*/;
P1 = ADDR(BUF);
P3 = ADDR(ARG2);
IF FLAG = SW2 THEN DO;
 FLAG=SW1;
END;
IF FLAG = SW1 THEN DO;
 PUT EDIT (REC) (COLUMN(2),A(80));
 GO TO GO_BACK;
END;
EOF:/* ALL OUTPUT WRITTEN */;
GO_BACK:RETURN;
END PLIPROG;

```



# Index

---

## C

control statements • 28, 29, 32  
    ADR • 28  
    ADR, pointed start • 29  
    blank • 29  
    dummy • 29  
    KEY • 28  
    KEY, direct read • 32  
    KEY, pointed start • 28  
conversion • 53, 55, 62  
    Julian date • 53  
    universal date • 55, 62  
conversions • 82, 84, 88, 91, 105  
    binary to alphanumeric • 105  
    binary to pack decimal • 105  
    doubleword binary to packed decimal • 91  
    floating point to packed decimal • 88  
    packed decimal to binary • 82  
    packed to zoned decimal • 84  
CULEMLIN • 128, 130, 136  
    page control • 128  
    page numbering, automatic • 128  
CULLUS11 • 54  
    error checking • 54  
    leap year • 54  
CULLUS11 procedure modules • 54  
    CULLUS11 • 54  
CULLUS12 procedure modules • 56, 57  
    CULLUS12 • 56  
    error checking • 57  
    multiple calls • 57  
CULLUS14 procedure modules • 58, 59  
    CULLUS14 • 59  
    Gregorian date • 58  
CULLUS15 procedure modules • 62, 64, 65  
    CULLUS15 • 62  
    error checking • 64  
    multiple calls • 64  
CULLUS22 • 66, 68, 69, 74  
    link edit • 66, 68  
    multiple ISAM files • 69  
    using multiple ISAM files • 68  
    z/VSE • 69  
CULLUS25 • 77, 78

    link edit • 77  
    manipulating key fields • 77  
    multiple VSAM files • 77  
CULSPAN • 20, 25, 26  
    source code • 20  
    spanned records, z/VSE • 20

## D

dump • 120, 122, 123  
    fixed-length records • 122  
    horizontal • 120  
    variable-length records • 122  
    vertical • 120

## F

floating point values • 91  
formatting • 120, 123, 126, 136  
    dump • 120  
    labels • 123  
    multiple lines • 126  
    VSAM records • 136

## I

input modules • 20, 26  
    CULLVSAM • 26  
    CULSPAN • 20

## L

link edit • 42, 66, 68, 69, 77, 146, 148  
    CULLUS22 • 68  
    CULLUS22 • 66  
    CULLUS22 • 69  
    CULLUS25 • 77  
    user-written modules • 42, 146, 148

## O

output modules • 119, 120  
    general discussion of • 120  
    summary of • 119  
    syntax • 120

## P

password • 28, 29, 31, 32  
    password omission • 29, 32

---

- PW= • 28
- procedure modules • 41, 42
  - argument sequence • 41
  - general discussion • 42
  - multiple • 42
  - numeric arguments • 41

## S

- sourcecode • 20, 69
  - CULLUS22 • 69
  - CULSPAN • 20
- syntax • 20, 27, 40, 43, 44, 66, 75, 79, 81, 82, 83, 84, 85, 86, 88, 89, 92, 93, 94, 96, 100, 101, 104, 105, 107, 110, 113, 114, 116, 117, 119, 120, 121, 122, 124, 126, 128, 136, 138, 139, 141
  - CULEDUMP • 121
  - CULELABL • 124
  - CULEMLIN • 128
  - CULEPOWR • 138
  - CULEVSAM • 136
  - CULLUS00 • 43
  - CULLUS01 • 44
  - CULLUS22 • 66
  - CULLUS25 • 75
  - CULLUS29 • 79
  - CULLUS31 • 81
  - CULLUS33 • 83
  - CULLUS34 • 85
  - CULLUS35 • 86
  - CULLUS36 • 89
  - CULLUS37 • 92
  - CULLUS40 • 93
  - CULLUS43 • 94
  - CULLUS45 • 96
  - CULLUS46 • 101
  - CULLUS48 • 104
  - CULLUS50 • 105
  - CULLUS53 • 107
  - CULLUS62 • 110
  - CULLUS64 • 114
  - CULLUS99 • 117
  - CULLVSAM • 27
  - CULSPAN • 20
  - input modules, general • 20
  - output modules, general • 120
  - procedure modules, general • 40
- system time and date • 47, 49, 53
  - Canadian format • 49

- retrieval of • 47

## U

- user modules • 12, 14
  - summary of • 14
  - types of • 12
- user modules, writing of • 146
  - general discussion • 146
- user-written modules • 154, 158, 160, 162, 163, 164, 168, 170, 171
  - Assembler • 148
  - Assembler input module • 158
  - Assembler output module • 170
  - Assembler procedure module • 163
  - COBOL input module • 154
  - COBOL output module • 168
  - COBOL procedure module • 162
  - FORTTRAN • 150
  - FORTTRAN procedure module • 164
  - PL/I • 149
  - PL/I input module • 160
  - PL/I output module • 171
  - PL/I procedure module • 164
- use-written modules • 42
  - linkedit • 42

## V

- VSAM file • 26, 29, 74, 136
  - direct read • 26
  - pointed start • 26
  - random access of • 74
  - read from the beginning • 29
  - relative byte address (RBA) • 26
  - sequential read • 26
  - writing to • 136

## Z

- z/OS • 69
  - CULLUS22 • 69
- z/VSE • 69
  - CULLUS22 • 69